VERKTØY I SAMSPILL

En kvalitativ studie av forholdet mellom verktøy og smidige metoder i systemutvikling

Av Erlend Andresen

Masteroppgave ved institutt for informasjons- og medievitenskap

Universitetet i Bergen
Vår 2012

Innhold

Forord

Mitt bakteppe fra jobbsammenheng, høyskole og universitetet har vært med å forme min forståelse av hvordan et mikrosamfunn – et team – arbeider sammen for å løse praktiske oppgaver og imøtekommer krav innen tidsfrister igjennom samarbeid og problemløsning. Jeg ønsker å takke alle som har hjulpet meg opp og frem i løpet av studietiden og jobbutfordringer, spesielt de to siste årene på institutt for informasjons- og medievitenskap i Bergen.

Jeg ønsker å takke Donia Lina for korrektur og oppmuntring underveis i et noe hektiskt tidsforløp den siste tiden, og for avkobling når hun så at jeg trengte det mer enn meg selv. Jeg vil rette en takk til Solveig Bjørnestad for veiledning, gode råd og faglige samtaler som har hjulpet meg frem mot det endelige produktet denne oppgaven har resultert i.

Takk for alt 638, det har vært særdeles hyggelig å dele «hovedfagsleseplass» med dere. Takk til "helikoptermannen" Kjetil, Mr. Tan Sindre og "jeg har ti-tusen twitter-followers" -Regine som gjorde studietiden gøy med treffende gullkorn (som like så greit ikke blir videre referert her) og mengder av usaklige diskusjoner over inntak av Fjorland™.

En spesiell takk til informantene som tok seg tid til intervju. Foruten dere hadde denne studien ikke vært mulig.

Motivasjon

Jeg ble introdusert til smidig utvikling sommeren 2010 da jeg arbeidet som sommervikar hos Opera Software ASA. Utviklingslaget jeg utførte kvalitetssikring for benyttet seg av kanban, en type lean[1] fremgangsmåte. Jeg fikk erfare bruk av kanbantavle hvor de skrev ut oppgaver fra prosjektstyringsverktøyet JIRA, som de deretter hengte opp på veggen i løpet av det daglige møtet med prosjektdeltakerne. Jeg hadde ansvar for å se på ulike måter man kunne teste produktet, en nettløsning, på klient-siden og jeg fikk sammen med teamet oversikt over prosjektet via kanbantavlen som fungerte som en visualisering på det daglige fokuset i prosjektet.

Parallelt med nettløsningsprosjektet deltok jeg i et vedlikeholdsprosjekt, hvor vi kun brukte JIRA for å spore oppgaver gjennom ulike faser i utviklingen, uten bruk av tavle. Sett opp mot bruken av den fysiske kanbantavlen, var det mer klikking i nettleseren, da prosjektene var forskjellige med tanke på ressurser og antall oppgaver i omløp. I vedlikeholdsprosjektet ble det oftere rapportert feil på programvare, og vi var flere distribuerte personer som arbeidet på samme prosjekt.

Uten å reflektere over disse forskjellene på det tidspunktet, har jeg i dag mer tyngde til å kunne si noe om hvordan forskjellige prosjektstyringsverktøy fungerer med tanke på måten informasjonen presenteres. Den påfølgende høsten tok jeg faget «Advanced Topics in Software Engineering» som omhandlet ulike metoder for smidig utvikling, og som følgelig tok for seg utfordringer innenfor smidig utvikling i teori og praksis. Flere ganger utover høsten oppsto faglige diskusjoner om datastøttede verktøy og fysiske artefakter og deres funksjon som informasjonsradiatorer. I nysgjerrighet utformet jeg en prosjektskisse med fokus på bruk av prosjektstyringsverktøy og smidig utvikling, hvor koordinering, visualisering og gjennomføring av smidige metoder var vektlagt.

[1]Norsk: mager

1 Innledning

En av måtene for å minimere risiko i prosjektstyring av systemutvikling er å bruke ulike smidige metoder. Smidige metoder beskriver regler og praksiser for hvordan utviklingsteam kan samarbeide for å levere kontinuerlig verdi til kunde. Utfordringene ved å utvikle systemer kan sies å være mange, og systemutvikling avhenger i stor grad av kommunikasjon innad i prosjektteam. På et generelt grunnlag kan man si at et produkt er vellykket når produktet er levert i fungerende tilstand, til rett tid og avtalt kostnad (Royce, 1970, s.328). En av utfordringene er å minimere risikoen slik at produktet ikke medfører større kostnader for de aktuelle aktørene; kunden får kanskje ikke levert produktet som ble skissert; prosjekteier må kanskje ta støyten for et feilutviklet produkt og kostnader det medfører; overarbeidede utviklere opplever ikke eierskap til koden og mister troen på prosjektet.

I dag ser man en utbredt bruk av smidige metoder, nemlig metoder som setter personer og samspill fremfor prosesser og verktøy. Likevel ser man en utbredt bruk av, og ofte i distribuerte prosjekter, et stort behov for, nemlig verktøy. Denne oppgaven tar for seg fem ulike personers erfaringer og bruk av ulike typer verktøy i kombinasjon med smidige metoder. Dette for å undersøke nærmere hvordan verktøy påvirker organisasjonens praksis av smidige metoder gjennom analyse av kvalitative intervjuer.

Personer og samspill fremfor prosesser og verktøy

XP ble presentert som en av de første smidige metodene som blir beskrevet i detalj av (Beck, 1999; Beck og Andres, 2004). Praksisene i XP har mye til felles med fossefall- og spiralmodellen, men XP vektlegger praksiser og regler for hvordan utviklerne kan arbeide sammen fremfor prosesser. Spesielt ser man dette i praksisene par-programmering og «felles kode-eierskap», samt hvordan man presenterer status i prosjekter i en daglig «standup». Scrum (Schwaber, 2004; Schwaber og Beedle, 2001) går mer i dybden i beskrivelsen av aktørenes roller i prosjektet, og beskriver blant annet hvilke oppgaver de ulike rollene har i prosjektsammenheng. Opphavspersonene av de første smidig metodene (med flere) gikk sammen for å utforme fellestrekkene for smidige metoder og ble kalt «Manifestet for smidig programvareutvikling» (Beck, Beedle, Bennekum, Cockburn, Cunningham, Fowler et al., 2012):

> «**Personer og samspill** fremfor prosesser og verktøy,
> **Programvare som virker** fremfor omfattende dokumentasjon,
> **Samarbeid med kunde** fremfor kontraktsforhandlinger,
> **Å reagere på endringer** fremfor å følge en plan.
>
> Dette vil si: Selv om punktene til høyre har verdi, så verdsetter vi punktene til venstre enda

høyere»

Cockburn (2000) fant at det var mangel av fokus på personer i systemutvikling. Sammen med første punkt i det smidige manifestet, er det grunnlag for å hevde at metoder for systemutvikling har skiftet fra en plan-orientert metodologi, med fokus prosesser og verktøy, til fokus på personer i samspill.

Det er også verdt å nevne endringen i metodologienes[2] omfang: XP og Scrum forsøker å gi et sett av praksiser og regler som er mindre omfattende (ofte referert til som lette rammeverk) kontra de tidligere, tradisjonelle metodene (ofte referert til som tyngre rammeverk).

Cockburn så på vellykkede systemutviklingsprosjekter for å finne fellesnevnere mellom de, og fant blant annet at «Light processes are more often successful, and more importantly, the people of those projects credit the success to the lightness of the methodology» Cockburn (2003, s.49). Påstanden kan tolkes som at utviklere er mer positive til lette rammeverk som kan beskrives gjennom praksiser og regler.

Et prosjektstyringsverktøy holder oversikt over pågående arbeidsoppgaver, mens utviklingsverktøy har en teknisk støtte for utviklerne i selve programmeringsprosessen. Verktøy for å støtte opp om kontinuerlig integrasjon[3] (Continous Integration) og små produktslipp[3] (Small Releases) kan i mange tilfeller integreres opp mot prosjektstyringsverktøy. I litteraturen er slike verktøy omtalt som konfigurasjonsverktøy og prosessautomatiserte verktøy.

1.1 Problemstilling

Goth (2009) argumenterer for at første punkt i manifestet for smidig programvareutvikling er en motarbeidelse av økningen av prosjektstyringsverktøy. Tradisjonelle verktøy som ikke støtter opp om konkrete metodologier benyttes nemlig også i smidige prosjekter. Spørsmålet blir da hvordan disse verktøyene påvirker bruken av smidige metoder. Hvordan løser konsulentfirmaer utfordringer i det å kommunisere og koordinere via et datastøttet prosjektstyringsverktøy kontra fysiske artefakter som kan sies å være et hovedelement i smidige metoder? I distribuerte prosjekter er bruk av datastøttede prosjektstyringsverktøy kanskje den eneste mulige løsningen. Kan disse to ulike måtene å kommunisere på kombineres, eller er den ene måten å foretrekke?
Hovedproblemstillingen tar dermed for seg hvordan verktøy påvirker organisasjoners praktisering av smidige metoder, med fire underspørsmål:

Hvordan påvirker verktøy organisasjonens praksis av smidige metoder?

[2] Det finnes flere smidige metodologier, men XP og Scrum beskrives for deres relevans i denne studien.
[3] XP-praksiser som blir forklart i teori

1. Hvilke verktøy benyttes i konsulentfirmaer innen programvareutvikling?
2. Hvilke utviklerverktøy er nødvendige for smidige prosjekter?
3. Støtter funksjonaliteten til prosjektstyringsverktøy prinsippene for den aktuelle smidige metoden i prosjektet?
4. Blir datastøttede verktøy brukt som erstatning for fysiske artefakter, eller lever verktøyene side om side?

I denne kvalitative studien ser jeg på smidige metoder og bruken av prosjektstyringsverktøy, i form av digitale verktøy og bruk av tavler. Jeg ønsker å stille spørsmål til hvilken påvirkning prosjektstyringsverktøy har på smidige metoder. I tillegg ser jeg også på hvordan utviklerverktøy støtter opp om smidige metoder.

Jeg har intervjuet fem personer med erfaring i smidige metoder hvor jeg har tatt utgangspunkt i en ustrukturert intervjuform og brukt mal-analyse som verktøy for å finne relasjoner mellom teori og praksis. Innsamlet data inneholder samtaler om begreper knyttet til smidige metoder og bruken av verktøy i praksis.

1.2 Oppgavens struktur

Kapittel 2 presenterer aktuell teori for problemstillingen. Først vil ulike smidige metoder bli beskrevet og deres rolle innenfor programvareutvikling. Deretter følger et innblikk i empiri på, og informasjon om, ulike prosjektstyrings- og utviklerverktøy som benyttes i smidige programutviklingsprosesser. Kapittel 3 presenterer ulike metoder som kan benyttes for å få svar på problemstillingen. Videre følger kapittel 4 om forskningsdesign som forklarer ulike faktorer for valg av metode og måten jeg skal innhente og analysere data. Kapittel 5 beskriver prosedyrene som ble fulgt for datainnsamlingen, fra planlegging til gjennomføring. Analysen av innsamlet materiale blir presentert i kapittel 6 og diskutert i kapittel 7. Avslutningsvis trekkes det en konklusjon om oppgavens innhold og drøfting av videre arbeid i kapittel 8.

2 Teori

Hva var bakgrunnen for at de smidige metodene ble utviklet og ble populære? Aktører så at de kunne minimalisere risikoen i komplekse utviklingsprosjekter ved hjelp av en smidigere fremgangsmåte, hvor man reagerte på endringer underveis i prosjekter. Praksiser og regler for å reagere på endringer underveis var ikke i like stor grad fremhevet i tradisjonelle utviklingsmetoder som tok utgangspunkt i tunge, planorienterte prosesser. For eksempel var modellen til Royce (1970) utarbeidet på bakgrunn av datidens statlige kontraktbegrensninger. Boehm (1988) beskrev spiralmodellen som en iterativ og inkrementell måte å arbeide på som smidige metoder vektlegger i enda større grad. Cockburn (2000) studerte en rekke utviklingsprosjekter hvor han så på faktorer som påvirket prosjektutfallet på en positiv måte, og fant blant annet at lette prosesser ga oftere suksess. I tillegg fant Cockburn (2000) blant annet viktigheten av tett kommunikasjon mellom utviklere.

Beck (1999, s.3) nevner risikoer ved programvareutvikling kan få stor økonomisk og menneskelig innvirkning, og lister eksempler på slike risikoer: Produktslipp går ikke som planlagt og kunde får ikke produktet til avtalt tid; prosjektet kanselleres og tas ikke i bruk; høye driftskostnader som en følge av dårlig programvare eller dyre endringer i programvare satt i produksjon; produktet løser ikke oppgaver det er ment til å løse; overflødig funksjonalitet uten verdi for kunde. Smidige metoder forsøker å minimere de listede risikoene.

Schwaber (2004) beskriver Scrum som en metode for hvordan å takle kompleks systemutvikling ved å definere relativt enkle regler og praksiser, som fungerer som et rammeverk for systemutvikling i stedet for planorienterte prosesser. Cockburn (2000) hevder at uansett hvor effektiv en metodologi kan være, så krever den disiplin over tid som personer har en tendens til å forkaste: «a high-discipline methodology is fragile» (2000, s.51). Dyba og Dingsoyr (2009) hevder at smidige metoder blir oppfattet som positive blant personer i fagmiljøet, og at de fremste faktorene som tett kundekommunikasjon og utviklere som er fornøyde med eget arbeid og produktene de utvikler, medvirker til denne oppfatningen.

Teorikapittelet vil ta for seg sentrale begrep og beskrive litteratur vedrørende de mest populære smidige metodene per idag, som Scrum, XP og lean programvareutvikling (Dyba og Dingsoyr, 2009). Per i dag ser man lean, og kanskje spesielt kanban, blir anvendt i smidige prosjekter (Xiaofeng Wang, Conboy, og Cawley, 2012). Videre følger en gjennomgang i litteraturen omkring prosjektstyringsverktøy og empiriske funn, med spesielt fokus på hvilke problemer disse verktøyene løser for smidige prosjekter. Prosjektverktøy og utviklerverktøy integreres ofte, derfor er det viktig å ta for seg begge.

Iterativ utvikling og simulering av produktet

Royce (1970) presenterte et rammeverk som er grunnlaget for det som senere ble kjent som fossefallsmodellen. For eksempel understreker modellen at man bør vektlegge iterativt samspill, altså åpne for muligheten til å gå tilbake en fase for å rette på mangler eller feil som har kommet frem av prosjektets fremgang. I et annet eksempel understreker Royce (1970, s.334) «do it twice» som beskrives som en kort testutviklingsfase, eller en form for tidlig simulering av produktet, som utføres før storparten av arbeidet skrider frem. Dermed er risikoen tatt høyde for i videre arbeid, som et resultat av simuleringen (Royce, 1970).

Boehms spiralmodell, presentert i 1988, er utarbeidet med vekt på risikohåndtering i form av blant annet gjennomgående iterativ utvikling (Boehm, 1988). Begrunnelsen er at man kan forutse utfordringer ved prosjektet og produktet i en tidligere fase i prosjektet. Tolkninger av (Royce, 1970) sin modell har ettertid vært utydelige. Royce beskrev sin modellen i kontekst av seksti- og syttitallets statlige kontraktmodeller, men var i grunn alltid en tilhenger av iterativ, inkrementell, evolusjonær utvikling: «vi ser hint av iterativ utvikling, tilbakemelding (feedback), og tilpasning i Royce sin artikkel. Steget basert på iterativ tilbakemelding har gått tapt i de fleste beskrivelser av denne modellen» (fritt oversatt, Larman og Basili, 2003, s.48).

Fokus på dokumentasjon

Royce (1970) fremhever dokumentasjon som en av hovedpunktene gjennom fasene i modellen. Royce vier mye oppmerksomhet til beskrivelsen av hvor mye dokumentasjon som er nødvendig, og kommer frem til at dokumentasjonen må være omfattende for å ha styring på komplekse prosjekter. Begrunnelsen for dette er at dokumentasjonen *er* spesifikasjonen av produktet.

Cockburn (2000) trekker frem det manglende fokuset på menneskelige faktorer i tidligere metoder, der verken fossefallsmodellen eller spiralmodellen vektlegger «aktive komponenter i systemet», altså menneskene i prosjektet (Cockburn, 2000), og hevder at personer i prosjekter har evnen til å forutse «project behaviour and methodology success». For å si det på en annen måte hevder Cockburn at det hjelper lite å legge til grunn tung metodologi og dokumentasjonsorientert prosesser når menneskene i prosjektet er en primær ressurs for suksess Cockburn (2000).

2.1 Extreme Programming

XP er en samling av praksiser, verdier og prinsipper som har blitt utviklet på grunn av problemene forårsaket av de lange utviklingssyklusene fra tradisjonelle systemutviklingsmodeller.. «In short, XP promises to reduce project risk, improve responsiveness to business changes, improve productivity throughout the life of a system» (Beck, 1999, s.xvi). Endringene i tenkemåte som Royce og Bohem

presenterte gjennom sine modeller ser man videreført i XP. Hovedsaklig dreier det seg om å minimere risiko i prosjekter. Men XP er mer fleksibel i måten praksiser anvendes, hvor man ikke må forholde seg til én modell, men i stedet implementere XP-praksiser i ledd hvor det er nødvendig.

Beck (1999, s.54) viser til 12 praksiser ved smidig systemutvikling i XP:

- *The Planning Game* – hurtig avgjøre omfanget av neste produktslipp ved å kombinere forretningsprioriteter og tekniske estimater, som oppdateres etterhvert som planen ikke stemmer overens med virkeligheten.
- *Small releases* – sett raskt opp et enkelt system i produksjon, og gjennom korte sykluser utfør nye versjoner av produktslipp.
- *Testing* – utviklere skriver kontinuerlig enhetstester, som må kunne kjøre feilfritt for at utviklingen skal kunne fortsette. Kunde skriver tester (akseptansetester) som demonstrerer at funksjoner er ferdige.
- *Pair programming* – all produksjonkode er skrevet av to utviklere på en maskin.
- *Continous integration* – integrer og bygg systemet mange ganger om dagen, på hvilket som helst tidspunkt.
- *On-site customer* – inkluder en bruker i teamet som hele tiden er tilgjengelig for å svare på spørsmål fra teamet.
- *Coding standards* – utviklere skriver kode i tråd med regler som understreker «communication through the code», som kan forstås som veldokumentert, forklarende og ikke-duplisert kode.
- De andre praksisene er metafor, enkelt design, refaktorering[4], kollektivt (kode)eierskap[5], 40-timers uke.

«Early estimation is a key difference between stories and other requirements practices» (Beck og Andres, 2004, s.44). Brukerhistorier («stories») beskriver en oppgave på ett kort med tekst, eventuelt med en tegning og estimering. Hvis en utvikler har fått i oppgave å utvikle en brukerhistorie, står gjerne navnet til utvikleren på brukerhistoriekortet.

2.2 Scrum

Uttrykket «Scrum» blir beskrevet som en tilpasningsdyktig, rask og selvorganiserende systemutviklingsprosess (Schwaber og Beedle, referert i Abrahamsson, Salo, Ronkainen, og Warsta,

[4]Eng.: Refactoring
[5]Eng.: Collective ownership

2002). Begrepet Scrum kommer opprinnelig fra en strategi i rugby som går ut på å få en ball ute av spill inn i spill igjen ved hjelp av lagarbeid (Schwaber, 2004, s.142).

> «The main idea of Scrum is that systems development involves several environmental and technical variables (e.g. Requirements, time frame, resources and technology) that are likely to change during the process.» (Abrahamsson et al., 2002).

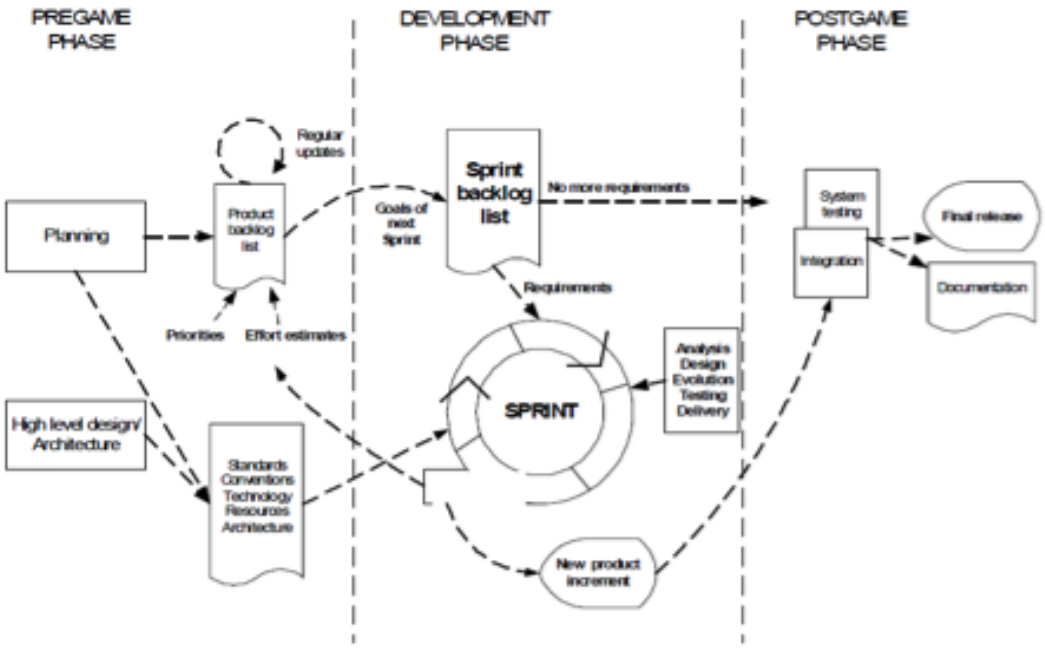
Siden miljøvariabler og tekniske variabler sannsynligvis forandrer seg i løpet av en prosjekt, legger Scrum vekt på tilpasningsdyktighet. Scrum legger dermed vekt på hvordan ulike roller fungerer sammen i et team, og skiller mellom de involverte aktører: De som må gjøre avansert teknologi om til funksjonalitet, og de som er tilskuere til utviklingsprosessen, som ikke nødvendigvis er ansvarlig ovenfor resultatet av prosjektet (Schwaber, 2004, s.7).

Scrum er i stand til å styre uforutsette endringer via Scrums kontrollmekanisme – gjennom empirisk prosesskontroll. «In the long run, making successful products the first time using empirical process control turns out to be much cheaper than reworking unsuccessful products using defined process control» (Schwaber, 2004, s.3). Ett eksempel på en slik empirisk prosesskontroll kan være kodegjennomgang, hvor en erfaren utvikler ser igjennom kode skrevet av en annen utvikler. Den erfarne utvikleren gir tilbakemeldinger i tråd med kodestandarder og anbefalt praksis. Kommentarer og forslag medfører at den andre utvikleren justerer koden sin deretter.

Schwaber beskriver skjelettet til Scrum:

> At the start of an iteration, the team reviews what it must do. It then selects what it believes it can turn into an increment of potentially shippable functionality by the end of the iteration. The team is then left alone to make its best effort for the rest of the iteration. At the end of the iteration, the team presents the increment of functionality it built so that the stakeholders can inspect the functionality and timely adaptations to the project can be made (Schwaber, 2004, s.6).

For å se flyten og sammenhengen i illustrasjonen forklares elementer som inngår i terminologien for Scrum.



Illustrasjon 1: Scrums tre faser (Abrahamsson et al., 2002).

Produktbaklogg

Produktbakloggen[6] er en liste bestående av estimerte, ugjorte oppgaver, funksjonelle, eller ikke-funksjonelle krav. Produkteieren, som representerer alle aktører i prosjektet, håndterer prioriteringen av produktbakloggen.

Sprint

En sprint er en tidsperiode på 30 dager (eller færre) hvor teamet utvikler oppføringer i produktbakloggen til et funksjonelt, leveringsklart produkt klart til utrulling. Et prosjekt består av flere sprinter.

Sprintbaklogg

Sprintbakloggen[7] er en liste bestående av de høyest prioriterte oppføringene i produktbakloggen som prosjekt-teamet definerer utover i sprinten. Ansvarlig person og estimert tid defineres for oppføringene i listen.

Nedbrennsdiagram

For å holde oversikt over fremgangen for sprinten, produktslippet eller produktet tegnes et nedbrennsdiagram[8], hvor gjenstående arbeid vises i y-aksen og tidsperioden (sprint eller sprinter) vises i x-aksen.

[6]Eng: Product backlog
[7]Eng: Sprint backlog
[8]Eng: Burndown Graph

Roller

Scrummaster påser at implementasjonen av Scrum fungerer. *Produkteieren* representerer alle aktører i prosjektet og håndterer prioriteringen av produktbakloggen. *Prosjektteamet* består av Scrummaster, produkteier og utviklere som har ansvaret for å utvikle programvare for hver sprint.

Møter

Daglig scrummøte[9] kalles ofte en «standup» hvor alle i prosjektteamet beskriver status og fremgang for sine arbeidsoppgaver. Eventuelle hindringer i arbeidet rapporteres til Scrummaster. Hver sprint begynner med et *sprintplanleggingsmøte*[10], hvor produkteier beskriver de høyest prioriterte oppgavene i produktbakloggen. Deretter diskuteres det mellom prosjektteamet og produkteier hvor mye som kan bli gjort i løpet av en sprint. Til slutt planlegger prosjektteamet hvordan de skal imøtekomme krav ved å planlegge og detaljere oppgaver inn i sprintbakloggen.

Et *sprint retrospektivmøte*[11] holdes i etterkant av en sprint for å finne ut hva som kan bli gjort bedre i neste sprint. Et eget *sprint evalueringsmøte*[12] holdes av prosjektteamet for demonstrasjon av ferdig funksjonalitet i produktet for produkteier og andre aktører (Schwaber, 2004, s.143-145).

2.2.1 De tre fasene i Scrum

De tre fasene i Scrum kalles for planleggingsfase (Pregame phase), spillfase (Development phase) og avslutningsfase (Postgame phase).

Planleggingsfasen

Planleggingsfasen innebærer to faser: planlegging og arkitektur/høynivådesign. Planleggingsfasen går ut på å lage en produktbaklogg som spesifiserer krav som er kjent.

«The Product Backlog list is constantly updated with new and more detailed items, as well as with more accurate estimations and new priority orders» (Abrahamsson et al., 2002).

I følge illustrasjonen ser man at produktbakloggen kan oppdateres før, men også i løpet av sprinter. Planleggingsfasen kan omhandle blant annet krav og definisjoner av prosjektteam, verktøy, risikovurdering og opplæringskrav. Planleggingen i arkitekturfasen er basert på elementene i produktbakloggen. For hvert nytt element som legges til i produktbakloggen må potensielle problemer og hindringer analyseres slik at prosjektteamet er klar over disse, noe som videre minimerer risikoen for prosjektet.

[9]Eng: Daily Scrum meeting
[10]Eng: Sprint planning meeting
[11]Eng: Sprint retrospective meeting
[12]Eng: Sprint review meeting

Spillfasen

Spillfasen beskrives som den smidige fasen i Scrum, også kalt «game phase», fordi det er i denne fasen uforutsette endringer kan skje og teamet må reagere på disse endringene. Sprinter planlegges og gjennomføres før produktet til slutt kan rulles ut, i avslutningsfasen.

Avslutningsfasen

I avslutningsfasen skal produktet ferdigstilles for utrulling. Hvis det er enighet mellom aktørene at ingen flere krav skal legges til, kan ingen flere (eller nye) oppgaver inkluderes i produktet. Oppgaver som inngår i avslutningsfasen er integrasjon, systemtesting og dokumentering (Abrahamsson et al., 2002, s.30).

2.3 Lean og kanban

«Eliminate waste, amplify learning, decide as late as possible, deliver as fast as possible, empower the team, build integrity, and see the whole» (Dybå og Dingsøyr, 2008, s.835).

Dette er de syv prinsippene for lean. Prinsippene kan ses i lys av hvordan smidige prosjektteam fokuserer på kontinuerlig forbedring av smidige utviklingsprosesser. I de senere årene har man sett at det smidige samfunnet tilpasser arbeidsmetoder etter lean prinsipper (Xiaofeng Wang et al., 2012). Lean kan anvendes i smidige metodologier på forskjellige måter. Den mest vanlige måten er å introdusere lean prinsipper i den eksisterende smidige metoden. To lean prinsipper som har fått mest oppmerksomhet er fokus på kundeverdi og eliminering av overflødige[13] elementer (Xiaofeng Wang et al., 2012).

«If you eliminate enough waste, soon you go faster than faster than the people who are just trying to go fast» (Beck og Andres, 2004, s.135). Beck refererer til hvordan Toyota Production System (TPS) fungerer i kortversjon. I følge TPS er den største «waste» av alle overproduksjon. Ved å bare ha akkurat nok arbeid i utvikling, vil man synliggjøre hvor en maskin som produserer en spesifikk bildel feile. Umiddelbart må feilen på den spesifikke maskinen i produksjonen utarbeides for å minimere kostnader.

Hvordan fungerer TPS, som er i utgangspunktet kommer fra bilproduksjon, i systemutvikling? Beck og Andres (2004) viser til hvordan utviklet programvare kan overproduseres, for eksempel store, hurtigvoksende kravspesifikasjonsdokumenter som blir utdaterte, eller at dokumentasjon som blir skrevet uten at noen oppdaterer eller leser den. Risikoen som følger ved å ikke få tilbakemeldinger på det man utvikler koster. Alle i et prosjektteam må reagere på endringer, gi tilbakemeldinger og

[13]Eng.: waste

tørre og si ifra hvis noe ikke fungerer. Slik opprettholdes kvaliteten på kildekoden som fører til mindre feilproduksjon. Nettopp disse elementene, blant flere, ser man videreført i XP. Poppendieck og Poppendieck (2003) beskriver i Tabell 1 hvordan lean kan overføres fra bilproduksjon til utvikling, med fokus på «muda», det vil si overflødige prosesser eller bortkastet ressurser.

The Seven Wastes of Manufacturing	**The Seven Wastes of Software Development**	**Syv «wastes» for programvareutvikling**
Inventory	Partially Done Work	Delvis ferdig arbeid
Extra Processing	Extra Processes	Ekstra prosesser
Overproduction	Extra Features	Ekstra funksjonalitet
Transportation	Task Switching	Oppgavebytting
Waiting	Waiting	Venting
Motion	Motion	Bevegelse
Defects	Defects	Feilvare

Tabell 1: Uttrykk overført fra produksjon til programvareutvikling - de syv "wastes."Tredje kolonne er lagt til med egen oversetting. (Poppendieck og Poppendieck, 2003, s.19).

Et lean konsept er kanban, ordet betyr ”signal”. Konseptet går ut på at det pågående arbeidet visualiseres og at flaskehalser synliggjøres i forbindelse med produksjon av oppgaver. Kanban deler likheter med Scrum og Scrumtavlen, som prioritert funksjonliste/oppgaveliste, men skiller seg likevel ut på to områder.

For det første er ikke kanban tidsbegrenset til iterasjoner, og hevdes å fungere bedre for vedlikeholdsprosjekter over utviklingsprosjekter: «the kanban approach suits particularly well to software maintenance or support type activities where uncertainty is higher than in normal development activities and change is more frequent than that allowed by agile iterations» (Xiaofeng Wang et al., 2012, s.11).

For det andre legger kanban opp til en begrensning i antall oppgaver i arbeid som synliggjøres på kanbantavlen, også referert til som CONWIP (Constant Work-In-Progress). Begresningen føres opp som et tall i hver kolonne for utviklingsfasene. For eksempel for et prosjekt med tre utviklere og en

tester har utviklingskolonnen en begresning på fem oppgaver i arbeid, mens testkolonnen har en begresning på to oppgaver i arbeid.

2.4 Prosjektstyringsverktøy

Teori som omhandler prosjektstyringsverktøy er ofte delt opp som fysiske artefakter på den ene siden og digitale prosjektstyringsverktøy på den andre. Fysiske artefakter brukes i denne studien som et begrep som omfavner all fysisk bruk av artefakter for å gjøre informasjon om et prosjekt synlig for aktører i et prosjekt, som av den grunn også kan betegnes som prosjektstyringsverktøy. Jeg beskriver i de underordnede seksjonene hva som kjennertegner de ulike måtene å radiere informasjon på.

2.4.1 Fysiske artefakter som prosjektstyringsverktøy

Bruken av fysisk artefakter, og håndtering av fysiske brukerhistorie-kort oppfordrer til «interaction and discussion between team members» (Sharp, Robinson, og Petre, 2009, s.115). Diskusjoner rundt tavlen retter fokus på essensielle endringer og hindringer i et smidig prosjekt.

Sharp et al. (2009) beskriver relevansen av fysiske artefakter i den sosiale konteksten som blant annet understreker den konstante bevisstheten av andres arbeid. Sharp referer til begrepet informasjonsradiator som viser endringer av informasjon over tid og som gir et oversiktsbilde på kort tid (Cockburn, referert i Sharp et al., 2009).

Sharp påpeker også ulemper med fysiske manipulering av brukerhistorie-kort, for eksempel at kort kan gå tapt, de kan ikke søkes igjennom eller deles mellom distribuerte prosjektteamdeltakere på en enkel måte (Sharp et al., 2009). På den andre siden hevder Beck og Andres (2004, s.45): «Every attempt I've seen to computerize stories has failed to provide a fraction of the value of having real cards on a real wall».

Planning poker

«Planning Poker is a widely-used technique for sizing user stories. It is a technique based on Delphi estimation that helps the team to size user stories using point values» (Chon, referert i Power, 2011, s.60-61). Molokken-Ostvold og Haugen (2007) ser på effekten av smidige estimeringsmetoder som planning poker, og innvirkningen det har på estimeringen hvis estimeringen utføres sammen i gruppe samlokalisert kontra inviduelt og «mekanisk». Personer har en tendens til å estimere optimistisk ved ansikt-til-ansikt-kommunikasjon, på individuelt nivå og gruppenivå, sammenlignet med en mekanisk kombinering på individuelt nivå. I min studie er dette interessant å se i lys av hvordan et verktøy som er ment å være fysisk overføres til individuelle avgjørelser uten fysisk

tilstedeværelse, da målet for planning poker i seg selv er ment å skape diskusjon rundt estimering av størrelsen for brukerhistorier. For eksempel fant Sharp et al. (2009, s.110) at fysiske tavler på generell basis ble brukt på følgende måte:

> At the start of an iteration, the team gathers around a large table and the cards are spread out for everyone to see. A discussion about estimating how much effort is required by individual stories, and deciding which stories can be tackled in the upcoming iteration ensues.

2.4.2 Digitale prosjektstyringsverktøy

Digitale løsninger gir muligheter i form av blant annet søkbare brukerhistorier, dynamiske nedbrennsdiagrammer og oppdaterte tavler for distribuerte prosjekter eller prosjekter som ser nytten av slike verktøy. Noen utfordringer følger likevel med overgangen fra fysiske artefakter til digitale løsninger:

> Any agile team looking to adopt digital support for their work will need to consider how to retain the advantages of the physical form while also benefiting from translation to the digital medium» (Sharp, Robinson et al. 2009).

Balansen mellom fordelene en digital løsning gir og hindre den også kan introdusere er viktig å ta høyde for ved valg og anvendelse av prosjektstyringsverktøy. Sharp et al. (2009, s.116) konkluderer med at fysiske notasjoner og egenskaper bør overføres til digitale verktøy for å ta høyde for de komplekse og sosiale forholdene som kreves for å fungere som et vellykket prosjektteam. Gjennom observasjon fant Sharp at fysiske og digitale verktøy kan kombineres, for eksempel ved å ta bilde av tavlen i avgjørende faser i prosjektet og legge ved bildene i den digitale løsningen (i det aktuelle eksempelet i en wiki-applikasjon).

Dubakov og Stevens (2008) beskriver i en subjektiv whitepaper for TargetProcess, Inc. hvordan den økende bruken av smidige metoder stiller høyere krav til effektive verktøy:

> «Rapid revolution demands new tools. There are many developers-focused tools appearing including JUnit, Eclipse, Cruise Control, etc. However, the project management community is less self-sufficient and unable to create new tools for the new methodology in their spare time. As a result, most companies are still using old-fashioned project management tools like MS Project or generic tools like spreadsheets for project planning and tracking.» (Dubakov og Stevens, 2008, s.3).

Dubakov og Stevens (2008) hevder at det er nærliggende at systemutviklere kontinuerlig forbedrer utviklerverktøy, i motsetning til «prosjektstyrings»-miljøet som har begrenset med tid og muligheter til å oppnå det samme for prosjektstyringsverktøy. Sett i lys av at de personene som i størst grad anvender prosjektstyringsverktøy ikke har de samme tekniske forutsetningene for å kunne

videreutvikle verktøy medfører at valget faller på tradisjonelle prosjektstyringsverktøy.

Ut ifra hva Hunt (2006) og Dubakov og Stevens (2008) beskriver som smidige verktøy ser man idag at flere verktøy har blitt utviklet med spesielt hensyn på smidige metoder. Hunt (2006) viser til verktøy som kan benyttes i prosjekter som bruker tradisjonelle metoder, eksempelvis Eclipse IDE, modelleringsverktøy, og versjonskontrollsystemer. I den kvantitative studien om programvarebedrifters bruk av verktøy og behovet for disse som støtter opp om smidige metoder fant Azizyan, Magarian, og Kajko-Matsson (2011) at 31 prosent av bedriftene benyttet både fysiske artefakter og prosjektstyringsverktøy. I tillegg fant de at smidige prosjektstyringsverktøy ble brukt i større grad i distribuerte kontra samlokaliserte prosjekter. Dette kan ses i sammenheng med at samlokaliserte prosjektteam ønsker å være smidig i tråd med praksiser for smidige metoder, med fokus på bruk av fysiske artefakter eller mindre avanserte, smidige verktøy. I distribuerte prosjekter hvor fysiske artefakter ikke kan benyttes, ser man tendensen i bruk av mer avanserte prosjektstyringsverktøy, som et alternativ fremfor avanserte, tradisjonelle verktøy. «Agile project management tools (...) aim in breaching communication distances between distributed teams» (Azizyan et al., 2011, s.37).

Mangler i funksjonalitet for smidige prosjektstyringsverktøy
Den mest savnede funksjonaliteten i de smidige prosjektverktøyene var støtte for rapportering: «The respondents mentioned the need for customizable, Agileoriented reports which are easier to use and more userfriendly» (Azizyan et al., 2011, s.36). I mangel på rapporteringsmuligheter ser man at personer med ulike roller ønsker ulik funksjonalitet. «The need for different views for different user roles» (Azizyan et al., 2011, s.36).

Kelter, Monecke et al (2003, s.421) sammenligner smidige prosjektstyringsverktøy med hvordan slike verktøy har blitt brukt i tradisjonelle utviklingsprosesser. Kelter påpeker faktorer som opplæringskostnad, kostnad, kompleksitet og mangel på fleksibilitet kan føre til at tradisjonelle verktøy ikke egner seg for smidige utviklingsprosesser. Kelter baserer sine funn på prosjekter som benytter XP. De generelle kravene til smidige verktøy oppsummeres i Tabell 2.

Til nå har begrepet smidige prosjektstyringsverktøy vært brukt som en paraply for ulike verktøy. Jeg vil videre plassere ulike eksempler på verktøy som har til hensikt å støtte opp om ulike smidige prinsipper.

Støtte for...	**Viktig hvis...**
Kommunikasjon, koordinasjon, samarbeid.	Store distribuerte lag, komplekse prosesser

Planlegging	Høy kvalitet og reliabilitet; gjennomgang (review) av eksterne referanser; heller uerfarne lagdeltagere.
Utvikling og dokumentasjon	En rekke dokument-typer, dokumenter er ferdige produkter; Strenge krav til kvalitet og konsistens
Konfigurasjonsstyring (Configuration Management)	Dynamisk utvikling i ulike versjoner/varianter; En rekke prototyper; mange (distribuerte) utviklere
Prosess-automatisering	Mange tidkrevende prosesser (generering, kontrollering), store systemer; heller uerfarne lagdeltagere

Tabell 2: Krav for smidige verktøy (Kelter et al., 2003).

Kelter hevder at «the lack of flexibility is a strong disadvantage of many heavyweight tools (...), forcing users to adapt their working practices to the tool – instead of the tool adapting to its users» (Kelter et al., 2003, s. 413). Siden 2003 er det naturligvis kommet flere verktøy som er utviklet til hensikt å nettopp være tilpasselig i henhold til valgt metode (Azizyan et al., 2011; Dubakov og Stevens, 2008; Goth, 2009), men likevel kan kravene i Tabell 2 diskuteres i lys av hvordan dagens verktøy fungerer.

Kommunikasjon, koordinasjon, samarbeid

Kommunikasjon, koordinasjon og samarbeid omkranser mange funksjoner som kan tilbys av prosjektstyringsverktøy. I tråd med Dubakov og Stevens (2008) er de letteste verktøytypene kontorprogramvare, som er lette å lære og bruke, samt fleksible. Kompleksitet øker med størrelse av prosjektteam og ved tilfeller av flere distribuerte prosjektteam bør prosjektstyringsverktøyet ha støttet for nettopp dette. Ved tilfeller av distribuert XP og «Scrum of Scrums» kan kommunikasjon, koordinasjon og samarbeid likevel utføres ved hjelp av telefonkonferanser eller e-postsystemer. Verktøyets kompleksitet kan være en ulempe i og med at det bidrar til å øke i takt med kompleksiteten i sammensetningen av prosjektteam. I tillegg nevner Kelter et al., (2003) slike verktøy for planlegging og milepæler, samt digitale tavler og samtidig (synkron) redigering i dokumenter.

Planlegging

Iterativ planlegging i smidige metoder er som tidligere beskrevet essensiell. Sharp et al. (2009, s.114) beskriver hvordan samlokaliserte prosjektteam gjennom iterative planlegging benytter et smidig planlegging, for eksempel «planning poker», og hevder at interaksjonen mellom deltakerne i spillet er et fundamentalt fysisk aspekt. Abstraksjonen som følger bruk av en digital løsning kan

dermed påstås å ha innvirkning for den påtenkte gevinsten.

Utvikling og dokumentasjon

Dokumenter i smidig utvikling kan ikke sies å være ferdige produkter, da dokumenter kan endres fra iterasjon til iterasjon. Jamfør manifestet for smidig programutvikling: «programvare som virker fremfor omfattende dokumentasjon» bør fokuset være på fungerende produkt ut til kunde. Smidige prosjekter legger mindre vekt på ferdig dokumentasjon. Bruken av brukerhistoriekort erstatter dokumentasjonen i den grad at oppgaver beskrives med tekst og/eller tegning. Annen relevant dokumentasjon trekkes gjerne ut i en type wiki (Kelter et al., 2003).

Kelter trekker frem CASE[14]-verktøy som i stor grad blir brukt i tradisjonelle utviklingsmetoder som også har en rekke funksjonsområder i smidige prosjekter. Kelter et al., (2003) beskriver at det bør være en balansen mellom hvor lett verktøyet er å bruke og i hvor stor grad verktøyet representerer påkrevd funksjonalitet. Formelle diagrammer og tegnede skisser for frihånd bør kunne brukes av alle prosjektdeltagere, og gjerne synkront. I tillegg bør verktøyene ha støtte for en ikke-tidkrevende synkronisering opp mot den oppdaterte kodebasen, slik at det modellerte materialet stemmer overrens med utviklet kode. På en annen side forteller Cockburn (2003, s.25) gjennom sine observasjoner en situasjon hvor en utvikler beskrev følgende situasjon: «we evaluated a bunch of CASE tools and decided that they would slow us down too much, so we didn't use any».

Konfigurasjonsstyring og prosess-automatisering

Kelter et al. (2003) trekker frem XP-praksiser som refaktorering, hyppig testing og kontinuerlig integrering som essensielle for en kontinuerlig oppdatert kodebase som kan bygges (build) til et fungerende produkt. For refaktorering trengs først og fremst en IDE som gir brukeren muligheter til å restrukturere og endre kildekoden. For å kunne gjennomføre hyppig testing av kildekoden (verifisere at det ikke er feil på kildekoden) behøver man kontinuerlig integrasjon, som kort fortalt bygger kildekoden etterhvert som utviklere sjekker inn kildekode-filer, eller ved et bestemt tidsintervall. Enhetstester skrives av utviklere og kjøres i forkant av innsjekking eller i løpet av integrasjonen mot produksjonskoden.

Verktøy som kategoriseres under begrepene utviklings – og dokumentasjonsverktøy, CASE-verktøy og konfigurasjons- og prosess-automatiserte verktøy fra litteraturen blir i denne studien definert under paraplybegrepet utviklerverktøy.

X. Wang, Maurer, Morgan, og Oliveira (2010) sitt utvalg av smidige verktøy tar tilsynelatende ikke

[14] Computer-aided software engineering. CASE-verktøy kan sies å være en foreldet term som omfavner mange tradisjonelle verktøy i forbindelse med utvikling av programvare, men verktøy i denne studien vil bli forankret i kategorier som blir brukt i mer moderne litteratur.

med i betraktning alle verktøy som støtter opp om smidig planlegging, slik at utvalget av verktøy ikke er direkte interessant for denne studien. Det som er interessant er selve kategoriseringen. Utvalget av verktøy gir likevel ett inntrykk av hva som menes med kategoribegrepene. De fleste verktøy, ikke begrenset til utvalget i Tabell 3, har funksjonalitet som overlapper i kategorier.

Informasjon om tilgjengelige verktøy

I forbindelse kategoriene som til nå er beskrevet (Kelter et al., 2003), (X. Wang et al., 2010), (Azizyan et al., 2011) for prosjektverktøy og utviklerverktøy ser man at flere eksplisitte, smidige verktøy finnes idag. Siden Kelter et al skrev om verktøy fra 2003, og X. Wang et al fra 2010, er det naturligvis kommet nye versjoner av eldre verktøy, samt nye alternativer av prosjektstyringsverktøy. Følgende utvalg av verktøy har kommet frem av tidligere forskning, i tillegg til verktøy som er tilgjengelig på internett. Det finnes ytterligere flere verktøy i tillegg til de som beskrives, og utvalget er ment som representative for kategoriene.

Planleggingsverktøy

X. Wang et al. (2010) studerte planlegging i smidige metoder og løsninger som støtter distribuerte prosjekter. Hyppigheten av planlegging i smidige metoder gjør at planleggingsfasen er essensiell. Gjennom observasjoner fant X. Wang et al. (2010) tre faktorer som på en positiv måte påvirker kvalitet for samlokaliserte planleggingsmøter: «Shared access index cards describing tasks; flexible use of index cards; easy interactions amoung meeting participants.». De tre punktene kan oppfylles ved å skrive ned brukerhistorier på kort hvor alle møte-deltagerne er presentert. Punktene kan videre hevdes å være like viktige for distribuerte prosjekter, men de kan i mange tilfeller ikke ivaretas på grunn av mindre effektive kommunikasjonskanaler, jamfør Cockburn (2000) som viser til ulike former for kommunikasjon. For eksempel er to personer forran en tavle en varmere kommunikasjonform enn to personer som kommuniserer via e-post.

Kategori	**Utvalg av verktøy**
Wiki	MASE, PMWiki, JSPWiki, MediaWiki
Web-basert applikasjon	Rally, VersionOne, ScrumWorks, XPPlanner, Mingle
Tavle-basert applikasjon	CardMeeting, Gluewiki, AgilePlanner, MASE, Mingle, Danube
Plugins for IDE	IBM Jazz, Jira + GreenHopper, ProjectCards
Synkrone, smidige planleggingsverktøy	DAP, CardMeeting

Bord-basert, smidig planleggingsverktøy	APDT

Tabell 3: Kategorier og utvalg av verktøy. (X. Wang et al., 2010, s.188)

2.4.3 Informasjonsdeling - wiki

Dencheva (2011) forklarer begrepet wiki som «easy to use hypertext systems that can be read and modified by its online users through their browsers» (Dencheva et al., 2011, s.2). X. Wang et al. (2010) klassifiserer wiki som en applikasjon for å sjekke prosjektstatus, oppdatere oppgavelister og se prosjektdeltageres arbeidsprogresjon, i tillegg til deling av informasjon innad i prosjektet, i form av innlegg av spørsmål og svar. Amescua (2010) beskriver hvordan de benyttet en wiki for intern læring for smidige prosesser: «to transfer and share the software process knowledge, and on the other hand, junior software engineers developed software products with a greater degree of independence.» (Amescua, 2010, s.434). Man kan dermed se på funksjonområdene til en wiki på flere måter, alt etter hvordan wikien er tilpasset prosjektet, eller hvordan den er tilpasset med hensyn på metodologi. Jeg vil videre bruke begrepet wiki som en plattform eller applikasjon for generell kunnskapsdeling i en organisasjon eller et prosjekt.

Confluence wiki

Et eksempel på en wiki er Atlassian Confluence (Confluence, 2012). Confluence er en nettbasert applikasjon og installeres gjerne på toppen av en eksisterende Atlassian-installasjon, men kan også installeres alene. Informasjon kan deles på tvers av prosjektteam, avdelinger og personer. Confluence egner seg for å spesifisere oppgaver for videre utviklingsoppfølging i JIRA, for eksempel bilder av prototyper og spesifikasjon av hva som skal utvikles, i tillegg til mer generell informasjonsdeling i prosjekter.

2.4.4 Prosjektstyrings- og problemhåndteringsverktøy

Azizyan et al. (2011) utførte en spørreundersøkelse om bruken av smidige verktøy i organisasjoner som anvender smidige metoder, og fant at smidige prosjektstyringsverktøy (mellom 65% og 85%) ble brukt i større grad kontra tradisjonelle prosjektstyringsverktøy (i underkant av 20%). Tradisjonelle prosjektstyringsverktøy støtter ikke nødvendigvis en spesifikk metode, selv om noen verktøy har funksjonalitet som inngår i tradisjonelle metodologier. Azizyan et al. (2011) fant at MS Project, som er et tradisjonelt verktøy som fortsatt anvendes i smidige prosjekter, var antatt å ha 8% av den totale verktøybruken. MS Project har blant annet støtte for planlegging av ressurser, arrangere oppgaver til å bli gjort innenfor tidsfrister, og behandle avhengigheter mellom hendelser (Dubakov og Stevens, 2008). Det tradisjonelle prosjektstyringsverktøy mangler for å støtte opp om

smidige metoder er i følge Dubakov og Stevens (2008, s.12) vedlikehold av produktslipp-baklogg, vedlikehold av produktslipp og iterasjoner, progresjonsrapportering, mulighet for å estimere på bakgrunn av progresjonsmåling, oppgavetavle.

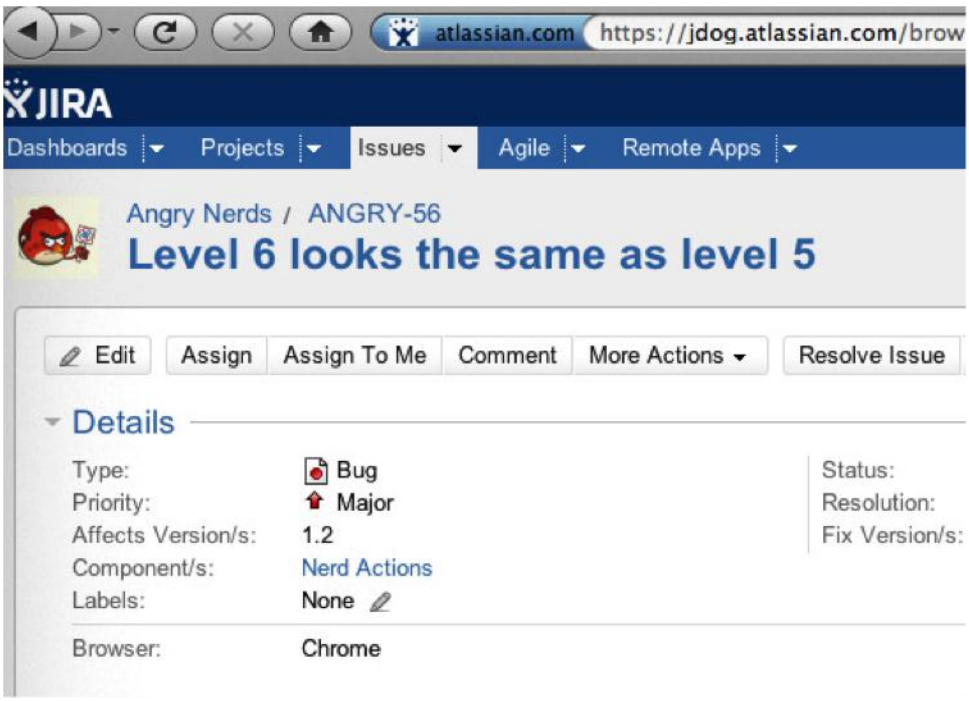
Andre prosjektstyringsverktøy, som for eksempel JIRA, åpner for muligheter til å tilpasse organisasjoners bruk av forskjellige smidige metoder ved hjelp av en tilleggsapplikasjon – GreenHopper. Prosjektstyringsverktøy som er utviklet med tanke på smidige metoder øker i volum (Goth, 2009) og de mest brukte av disse oppgir Azizyan et al. (2011) til å være Rally[15] med fem prosent. Selv om det finnes mange prosjektverktøy utviklet med spesielt hensyn på smidige metoder, anvendes likevel lette verktøy som fysiske artefakter og regneark[16] (ibid). For å vise hvordan prosjektstyringsverktøy fungerer vil JIRA beskrives mer i detalj, fordi det representerer den ikke-smidige delen av prosjektstyringsverktøy, samtidig som JIRA har støtte for tilleggsprogramvare, JIRA GreenHopper, som er utviklet med tanke på livssyklusen for smidige prosjekter, henholdsvis Scrum, lean og kanban.

Atlassian JIRA

På atlassian sine nettsider beskrives nettløsningen JIRA (JIRA, 2012) med stikkord som «Track, Plan, Analyze». JIRA håndterer prosjekter, feil, oppgaver, personer og kildekode, og gir muligheten til å styre prosjekter via nettleseren ved å generere innhold basert på aktiviteter utført av prosjektdeltakere. JIRA kategoriseres ofte som et oppgavehåndteringssystem[17] og Prause, Scholten, Zimmermann, Reiners, og Eisenhauer (2008) beskriver hovedformålet med verktøyet som «a bug-tracker in the field of software engineering addressing the tracing of error states and later the process of error correction» (Prause et al., 2008, s.154). JIRA åpner også for håndtering av flere fleksible funksjoner for store, samlokaliserte- eller distribuerte prosjektteam:

> «Jira enables the creation of workflows that map a specific issue's life cycle according to specific business processes. Different input fields per issue tracked by Jira as well as entire issue management workflows can be re-defined. Furthermore, the reporting functionality of this tool can be configured as well: notifications can be sent via email or RSS feeds if an issue gets into a specific state» (Prause et al., 2008, s.154).

[15]Rally software - http://www.rallydev.com/
[16]Omtrent 50% for samlokaliserte team og i underkant av 40% for distribuerte team.
[17] Termen oppgavehåndteringssystem og prosjektstyringsverktøy blir i denne studien brukt om en hverandre



Illustrasjon 2: Prosjektverktøyet Atlassian JIRA visualiserer en feilrapport (JIRA, 2012).

Skreddersydd livssyklus for forskjellige typer av oppgaver samt konfigurerbare rapporteringsmuligheter kan hevdes å være effektive funksjoner for organisasjoner som ønsker å tilpasse arbeidsflyten i utviklingsprosesser. Prause et al. (2008) viser til muligheter i JIRA for å automatisk genererer rapporter og gi tilgangsrettigheter for brukere, team og prosjektroller: «This allows to involve very different kinds of stakeholders in the requirements process» (ibid, s.154). Prause et al. (2008) fokuserer i den aktuelle studien på kravspesifikasjonsprosessen rundt systemutvikling på generell basis, og nevner i utgangspunktet grunnleggende funksjoner for JIRA uten hensyn til utviklingsmetodologi. Fleksibiliteten til JIRA er ikke begrenset til å konfigurere arbeidsflyten for oppgaver og prosjekter, da JIRA har støtte for installasjon av innstikk for å endre visningssider eller strukturere informasjon på alternative måter. Den smidige tilleggsprogramvaren Atlassian GreenHopper er et eksempel på et slikt innstikk.

Atlassian GreenHopper – Smidig prosjektstyring

GreenHopper tilbyr funksjonalitet for både Scrum og lean-baserte prosesser som for eksempel kanban. GreenHopper kan anvendes til å visualisere smidige prosjekter på flere måter. Sarkan, Ahmad & Bakar (2011) beskriver fire ulike tavler i GreenHopper: Planleggingstavle – håndterer utviklingsoppgaver; oppgavetavle – visning av arbeidsflyt som simulerer en fysisk tavle, med mulighet for å dra-og-slipp mellom utviklingsfaser, som for eksempel «To do», «In progress» og «Done»; diagram-tavle – muligheter for visualisering av oppgaver ved hjelp av diagrammer, som

for eksempel kumulativt diagram og nedbrennsdiagram; produktslipp-tavle – visning overoppgaver som skal komme med i neste produktslipp/iterasjon. viser hvordan et nedbrennsdiagram kan gi oversikt over progresjonen.

Illustrasjon 3: Nedbrenndiagram for Atlassian GreenHopper (GreenHopper, 2012)

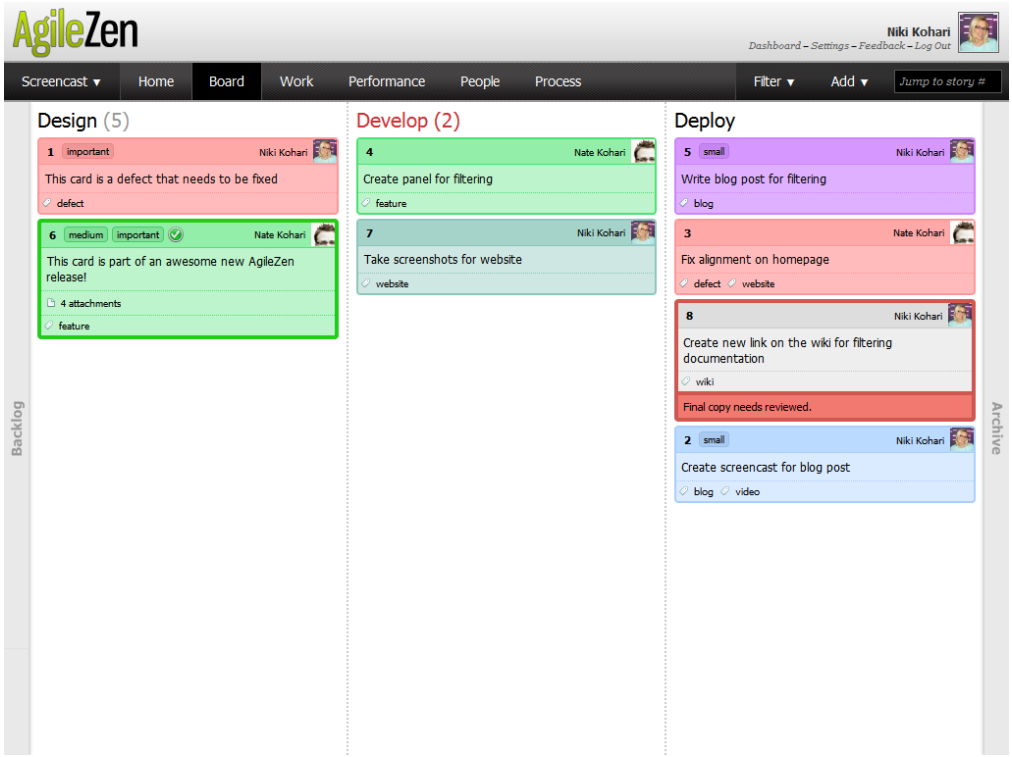
Søk og behandling av kode – Atlassian FishEye og Atlassian Crucible

Atlassian beskriver FishEye[18] som kildekode-surfing i nettleseren, som gjør det mulig å søke i kildekoden og bla igjennom innsjekket kode, filer, versjoner eller personer (JIRA, 2012). Sammen gir FishEye og Crucible[19] et alternativ til lesing/gjennomgang av innsjekket kode i en IDE. Siden det integreres mot JIRA, har man mulighet til å se hvilken JIRA-bruker/utvikler som endrer[20] kildekoden i ulike grener[21].

AgileZen – lettvekts prosjektstyring

AgileZen er for meg bekjent ikke beskrevet i litteraturen, men på nettsiden[22] til AgileZen presenteres den enkle nettapplikasjonen i hovedsak som en visualisering av en fysisk artefakt, i form av en Scrumtavle, kanbantavle eller andre generiske visningstyper som smidig/lean arbeidsflyt.

[18]Atlassian FishEye - http://www.atlassian.com/software/fisheye/overview
[19] http://www.atlassian.com/software/crucible/overview
[20]Eng.: Commits
[21]Eng.: Branches
[22] AgileZen - http://www.agilezen.com



Illustrasjon 4: Et kanbanprosjekt i AgileZen[23]

I Illustrasjon 4 viser en kanbantavle med begrensninger i antall oppgaver i arbeid for designkolonnen (5) og utviklingskolonnen (2). Kolonner og begrensninger kan slettes, endres og legges til i forhold til hvordan man ønsker at arbeidsflyten og livssyklusen for oppgavene i prosjektet skal være. Detaljer som vises for en oppgave er estimert tid, oppgavetaker, beskrivelse og type, samt en fargekode som gjenspeiler type.

2.5 Konfigurasjons- og prosessautomatiserte verktøy

Hunt (2006, s.217) ser på verktøy for å støtte opp om smidig utvikling, og beskriver ut fra egen erfaring hva som er viktig for at en smidig utviklingsprosess skal kunne følges:

- Det bør være mulig for å refaktorere programvare på en enkel måte, noe som kan gjøres i en IDE[24].

- Det bør være mulig å modifisere programvare, uten å på forhånd vite om noe går galt. Hvis noe går galt, skal det være en mulighet for å stille tilbake programvare på et fungerende tidspunkt. Her foreslås det å benytte et lett modelleringsverktøy for å hjelpe til med smidig modellering. Et eksempel på et slikt verktøy kan være en plugin[25] for en IDE.

[23] http://www.agilezen.com/content/css/content/images/external/kanban-project-management-lg.png
[24] Integrated Development Environment
[25] Norsk: innstikk

- Et bygg-verktøy for å bygge systemet når det er nødvendig. Et eksempel på et byggverktøy kan være ANT.
- Et versjonsstyringsverktøy som behandler hyppige og raske endringer på kildekoden, og gi muligheten til å rulle tilbake om nødvendig. Eksempler på slike verktøy kan være CVS.
- Et testrammeverk for utføring av enhetstester (tester skrevet av utviklere). Et eksempel på et slikt verktøy kan være JUnit.

Listen viser til utviklerverktøy som anvendes for å utvikle programvare. Disse nevnes av Hunt til å kunne anvendes i en smidig utviklingsprosess, selv om disse verktøyene ikke er ekslusivt smidige.

2.5.1 Byggverktøy

Ant[26] er et fleksibelt Javabibliotek og et kommandolinjeverktøy som kan bygge systemer og dets avhengigheter. Ant bruker XML[27]-filer til å håndtere byggeprosessen, og på bakgrunn av hva som er definert i byggfilen kan Ant håndtere et bygg som automatisk henter kildekode fra versjonshåndteringssystemer, kompilere javakode, opprette, kontrollere og produksjonssette Java-filer (jar[28], WAR, EAR), samt automatisk generere dokumentasjon (Javadoc).

Kontinuerlig Integrering – Atlassian Bamboo

Bamboo kjører enhetstester og bygger kildekode etter at en utvikler har endret sjekket inn kode. Verktøyet kjøre automatiserte tester og rulle ut programvare automatisk. Sigerid & Baggioloni (2011) beskriver hovedfunksjonen til Bamboo:

> «Whenever a developer commits changes to the source code repository, [bamboo] checks out the new sources, compiles them and runs the unit tests. It then does the same with all dependent projects, in a cascading way, to assure that everything still compiles and all the unit tests still succeed.» (Sigerud og Baggiolini, 2011, s.1214).

Bamboo kategoriseres som et kontinuerlig integrajonsverktøy og støtter blant annet opp XP-praksisen kontinuerlig integrasjon.

Hudson[29] er et kontinuerlig integrasjonsverktøy som overvåker gjentakende prosesser. For eksempel kan Hudson kjøres for bygging og testing av programvare, som Hudson på sine nettsider hevder å lette arbeidet for utviklere med å integrere endringer i prosjekter. Hudson kan også overvåke

[26] Apache Ant - http://ant.apache.org/
[27] Extensible Markup Language
[28] Filnavn for javaarkiv
[29] Hudson - http://wiki.eclipse.org/Hudson-ci/Meet_Hudson#What_is_Hudson.3F

resultater av eksterne prosesser. Resultater kan presenteres i form av RSS[30], e-post, eller via lynmeldingstjenester[31]. Hudson integrerer også med JUnit.

2.5.2 JUnit – testrammeverk

> «JUnit is an example of a standardised, well-established testing framework (…) JUnit provides a simple way to explicitly define repeatable unit tests and test suites. It is written in Java and so can be easily integrated into any Java development. It is also possible to integrate JUnit into ANT to automate regression test suites» (Hunt, 2006, s.227-228).

JUnit kan blant annet integreres i Eclipse for å enhetsteste kildekoden og automatisere regresjonstester. JUnit ble skrevet av Erich Gamma og Kent Beck (samme forfatter som XP) og er modellert på xUnit testrammeverket (ibid).

2.5.3 Versjonskontrollsystemer

«Version control systems are essential for co-ordinating work on a software project» (de Alwis og Sillito, 2009, s.36). Systemer for versjonhåndteringsystemer (VCS)[32] har lenge vært nødvendig for å holde orden på kildekode i systemutvikling. Imidlertid ser man at sentraliserte VCS som CVS og Subversion[33] blir erstattet av en ny generasjon av VCS, nemlig distribuerte VCS (DVCS), som for eksempel Git og Mercurial. DVCS gjør det unødig å ha en sentralisert hovedlager[34] for kildekode, og gir utviklere mulighet til full skrivetilgang i sitt eget personlige lager. Det hevdes at DVCS på en enklere måte håndterer grener[35] og gjentakende sammenslåing[36] av kode, og kan derfor være velegnet for distribuert utvikling (de Alwis og Sillito, 2009). På Gits nettside[37] nevnes fordeler for distribuert versjonshåndtering, blant annet forenklet kontekstbytting som gjør det lettere å arbeide mellom produksjonskode og egen gren, samt enklere oppstykking av grener. Effektiviteten hevdes også å være forbedret over Subversion: «Git is one or two orders of magnitude faster than SVN, even under ideal conditions for SVN»[37].

[30] Really Simple Syndication
[31] Eng.: Instant Messaging
[32] Eng.: Revision/Version Control System
[33] Også kalt SVN
[34] Eng.: Main repository
[35] Eng.: Branches
[36] Eng.: Merging
[37] http://git-scm.com/about

3 Metode

Med en problemstilling som stiller spørsmål til organisasjoners bruk av verktøy, ble det viktig å finne en metode for å undersøke ikke bare hvilke verktøy som blir tatt i bruk og hvorfor, men mest sentralt er spørsmålet om hvordan de verktøyene som brukes påvirker systemutviklingen i organisasjonen. Dermed var det sentralt å ta i bruk en kvalitativ forskningsmetode som kunne gå i dybden og som kunne være fleksibel ovenfor det informantene opplevde som sentrale punkter.

Kvalitativ forskning har ikke en endelig mal eller struktur for gjennomføring av intervju. Dermed blir måten man velger å gå frem på tilpasset enkeltvis for hver studie. Fokuset bør være på å dekke alle interessante emner relatert til problemstillingen. Når problemstillingen er definert for studiet, nevner King nevner i Symon og Cassell (1998, s.19) tre viktige synsvinkler som bør komme frem i en intervjuguide; direkte forskningsrelaterte spørsmål til problemstillingen; egne erfaringer og kjennskap til problemstillingen som forsker; ustrukturerte samtaler med personer som har erfaring innen forskningsfeltet. Kvalitativ forskningsmetode er grunnlaget for min metode, hvor jeg forsøker å tolke subjektive meninger fra et utvalg av informanter. «Det finnes få standardregler eller felles metodologiske konvensjoner i de kvalitative forskningsmiljøene» (Kvale, 1997, s.27). Intervjuprosessen bør dermed være åpen for ulike innvendinger og synspunkt informanter besitter.

3.1 Grounded theory

«One does not begin with a theory, then prove it. Rather, one begins with an area of study and what is relevant to that area is allowed to emerge» (Strauss og Corbin, 1990, s.23). Nærmere forklart samler en først data for deretter å se på hvilke aspekter som er relevante. En mer presisert forklaring kommer frem i Bryman (2008, s.541) som påpeker at grounded theory ikke er en teori, men en fremgangsmåte for å skape teori ut fra data.

Et av de mest sentrale aspektene av utviklingen av grounded theory er selve kodingen av datamaterialet. Ved å anvende koding på datamaterialet kommer man etter hvert frem til en kategorisering av data. Dette gjøres ved navngiving av forskjellige aspekter for problemet (open coding) for så å lenke nye forhold mellom kategoriene (axial coding), for til slutt å knytte nye forhold (selective coding) mellom kategorier man finner i forholdene mellom kategoriene (Strauss og Corbin, 1990).

3.2 Det kvalitative forskningsintervju

Bryman (2008, s.435) hevder at det kvalitative forskningsintervjuet holder seg til informantens syn på fenomenet i større grad enn det strukturerte forskningsintervjuet, i tillegg informanten formening om hva som er relevant for det aktuelle temaet. Det kvalitative forskningsintervjuet oppfordrer dermed til å tilpasse retningen av studiet mot problem reist av informanten.

> The qualitative research interview is most appropriate:
>
> Where a study focuses on the meaning of particular phenomena to the participants.
>
> Where individual perceptions of processes within a social unit – such as a work-group, department or whole organization – are to be studied prospectively, using a series of interviews. (...)». (Symon og Cassell, 1998, s.16).

Det første punktet beskriver studiets fokus på et spesielt fenomen. I denne oppgaven skal motsetninger i det første punktet i det agile manifest om personer og samspill fremfor prosesser og verktøy, og trenden, og gjerne nødvendigheten av slike verktøy i programutvikling. I begynnelsen av denne oppgaven var det nødvendig å drøfte hvilke metoder som kunne belyse viktige aspekter for nettopp dette fenomenet. Det andre punktet peker på at kvalitative forskningsintervju egner seg for en arbeidsgruppe, avdeling eller en hel organisasjon. Denne oppgaven tar for seg et utviklingslag sett gjennom ett eller flere prosjekter, og man har dermed belegg for å si at oppgaven ligger innenfor denne kategorien.

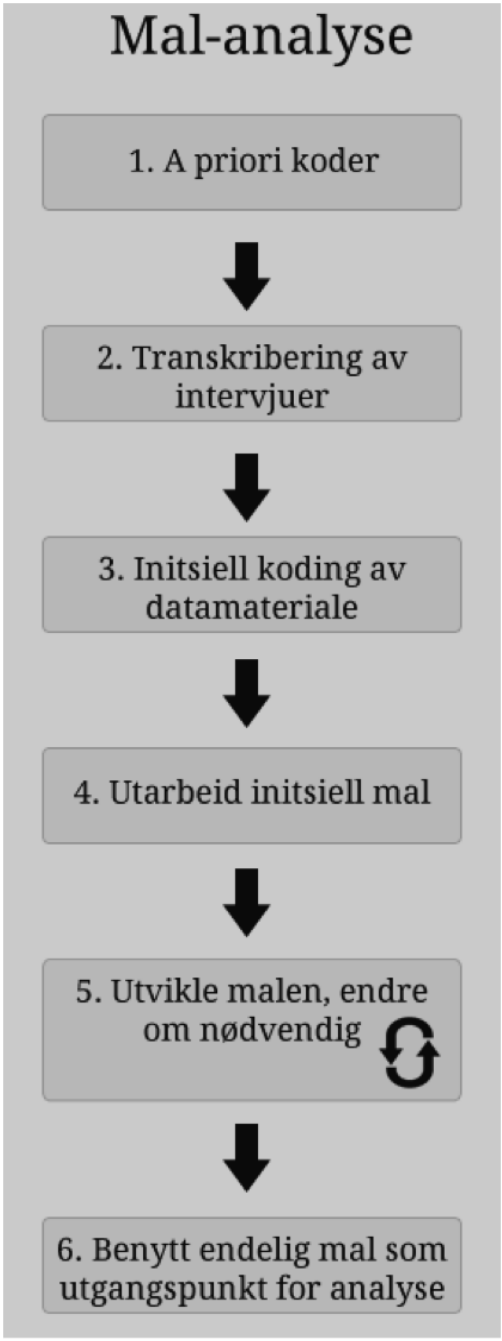
3.3 Mal-analyse

Mal-analysen følger ett sett av prosedyrer som til slutt munner ut i en mal for å analysere det totale datasettet. Denne formen for analyse kan benyttes hvor en innehar datamateriale i form av transkriberte intervju og hvor en videre ønsker å kartlegge funn. Mal-analyse kan benyttes som analyseverktøy for både strukturerte og mindre strukturerte intervju. King (2011) introduserer mal-analyse som: «a particular way of thematically analysing qualitative data. The data involved are usually interview transcripts, but may be any kind of textual data» (King, 2011).

Mal-analyse forsøker å hjelpe forskeren i å tematisere innhold ved å utarbeide en mal som er ledet ut fra kodingen av datamaterialet:

> «Template analysis involves the development of a coding "template", which summarises themes identified by the researcher(s) as important in a data set, and organises them in a meaningful and useful manner» (King, 2011).

A priori-koder for mal-analyse



Illustrasjon 5: Oversikt over mal-analyse

A priori-koder plasseres gjerne som høyeste noder[38] i et kodehierarki som utgjør utgangspunktet for første koding. Disse toppnodene får, i løpet av utviklingsprosessen av malen, flere underkategorier slik at malen bygges på etter hvert. A priori-koder kan ses på som hensiktsmessige hvis intervjuene svarer til disse. Hvis ikke, kan disse utelates i det videre arbeidet med materialet:

> «Analysis often, though not always, starts with some *a priori* codes, which identify themes strongly expected to be relevant to the analysis. However, these codes may be modified or dispensed with altogether if they do not prove to be useful or appropriate to the actual data examined» (King, 2011).

A priori-koder er kun en rød tråd for videre analyse. Fordelene er å bevisstgjøre forskeren på hva man leter etter i datamaterialet. En fallgruve er å gå tilbake i samlede data for kun å gjøre funn blant a priori-kodene: «Ved å fokusere på data som passer a priori-koder kan man overse materiale som ikke relateres til disse» (fritt oversatt, King, 2011). Mal-analyse gir mulighet for gå tilbake i temaene for å legge til relevante, eller fjerne irrelevante tema. Tankegangen er at et kvalitativt forskningsprosjekt vil endre seg i takt med bredere forståelse, og at det i tillegg skal kodes med oppmerksomhet mot relevant problemstilling. Det er derfor viktig å ikke avskrive temaer tidlig i analysefasen (Symon og Cassell, 1998).

Transkribering av intervju

Gjennom transkribering av intervjuene gjør man seg enda bedre kjent med materialet og stiller med en viss forkunnskap til koding og analyse. Fremgangsmåten i mal-analyse fremmer en iterativ prosess hvor analysen dras tidligere inn i kode-arbeidet, og den forkunnskapen forskeren sitter på kan være nyttig for å opprette tema som videre kan fungere som merkelapper for innholdet som skal kodes. Mal-analyse oppfordrer til en kort beskrivelse fra forskerens ståsted om hvordan materialet i intervjuet kommer frem i lys av forskningen som helhet.

[38] Et blad i en data/tre-struktur

Første mal

«When producing your initial template, it is important to cover the main thematic areas emerging from your preliminary analysis» (King, 2011). Her referes de «preliminære analysene» til de kortfattede sammendragene som anbefales til å skrives ned for hvert intervju i etterkant av transkripsjonsprosessen. Dette kan hjelpe med å få en bedre oversikt over datamaterialet på et tidlig tidspunkt i analysefasen.

Videre vil det gjerne oppstå spørsmål omkring *når* man skal avgjøre om den første[39] malen er ferdig. King (2011) beskriver at den første malen kan være ferdig etter koding av ett intervju eller mesteparten av intervjuene, eller på stadiet mellom en av disse ytterpunktene. Hvis studiet er av fenomenologisk type anbefales det ferdigstille den første malen på et tidlig stadiet, i motsetning til hvor studiet forsøker å svare på et spesifikt, teoretisk spørsmål. I det sistnevnte tilfellet kan det være hensiktsmessig å vente til en har kodet alle intervjuene for å unngå forutbestemmelser som videre vil påvirke hvordan den første malen vil se ut til slutt.

Utvikling av mal

Utvikling eller omgjøring av en mal kjennetegnes ved en iterativ prosess, hvor man benytter den første malen på hele datasettet. Der hvor man finner det nødvendig legger man til tema, og der hvor man ser at et tema kan fjernes på grunn av irrelevans opp mot problemstillingen, kan dette gjøres. Alle endringer som gjøres underveis må også tas hensyn til i de foregående iterasjonene. Til slutt sitter man igjen med en sammenfattet, endelig mal som tar for seg aktuelle kategorier og tema for alle intervjuene, som senere kan benyttes til analyse og diskusjon (King, 2011).

Oppsummering av mal-analyse og endelig utarbeidet mal

« A common mistake made by inexperienced qualitative researchers is to think that interpreting findings is simply a matter of summarising the interview contents indexed under each theme» (King, 2011). I følge King (ibid) bør endelig utarbeidet mal representere hvilke kategorier og tema fungere som et verktøy for å hjelpe forskeren i bearbeidelsen av analysen. En balanse mellom prioritering og åpenhet må ligge til grunn. Den endelige malen vil tvinge forskeren til å revurdere problemstillingen hvis kategorier så langt i forskningsprosessen ikke er beskrevet (ibid).

3.4 Evaluering av kvalitativ metode

Studiets reliabilitet

Bryman (2008) beskriver hvordan ekstern reliabilitet referer til hvor lett studien kan bli reprodusert, noe som sjeldent kan utføres uten at den «nye» forskeren har samme sosial rolle som den

[39] Eng.: Initial template

opprinnelige forskeren. Dette begrunnes med at man ikke kan «fryse» den sosiale situasjonen for den opprinnelige studien (Bryman, 2008, s. 376). Intern reliabilitet refererer til hvor mange observatører som er tilstede under intervjuene. Desto flere som ser og hører informantens syn på problemet gir studien en bedre intern reliabilitet (Bryman, 2008).

Studiets validitet

Studiets interne validitet kan måles i hvor lenge observatøren befinner seg i det sosiale miljøet det forskes på. Dette gjelder særlig for etnografiske studier hvor forståelsen mellom konsepter og observasjoner valideres over en lengre periode. Intervjuer kan i denne sammenheng hevdes å ha en lavere grad av intern validitet enn for eksempel en casestudie. Bryman (2008) forklarer ekstern validitet som i hvor stor grad studiet kan generaliseres på tvers av sosiale forhold. En studie med et lite utvalg, som kvalitative metoder gjerne har, har dermed en noe lav grad av ekstern validitet i utgangspunktet (ibid, s.376).

Generaliserbarhet

Kvalitative data er ofte vanskelig å generalisere fra de enkelte tilfellene og informantene til befolkningen. Det er derfor mer aktuelt i denne studien å se på forholdet mellom teori og eksempler som kommer frem i studien: «It is the quality of the theoretical inferences that are made out of qualitative data that is crucial to the assessment of generalization» (Bryman, 2008, s. 392). Analysert data kan settes i kontekst av den foreliggende teorien, slik at man eventuelt ser tidligere påpekte likheter i teorien. Det vil si at man ikke generaliserer på tvers av ulike informanters syn på et fenomen, men at man tar for seg eksempler som fremkommer av datamaterialet og sammenligner med teori.

4 Forskningsdesign

Problemområdet er definert som et bredt spekter av både samlokalisert og distribuert programutvikling. Dagens konsulentfirmaer bruker digitale og fysiske prosjektstyringsverktøy for å løse utfordringer i tilknytting til kommunikasjon og koordinasjon i prosjekter. Den overordnede spørsmålet i denne studien dreier seg om hvordan verktøy påvirker organisasjoners praktisering av smidige metoder. For å undersøke dette problemområdet må forskningsdesignen legge opp til å til sammen svare på de fire underspørsmålene:

1. Hvilke verktøy benyttes i konsulentfirmaer innen programvareutvikling?
2. Hvilke utviklerverktøy er nødvendige for smidige prosjekter?
3. Støtter funksjonaliteten til prosjektstyringsverktøy prinsippene for den aktuelle smidige metoden i prosjektet?
4. Blir datastøttede verktøy brukt som erstatning for fysiske artefakter, eller lever verktøyene side om side?

4.1 Fokus på forskningsspørsmålet

Problemstillingssettet forsøker å gjøre rede for forholdet mellom verktøy og smidige metoder i organisasjoner. Det er et åpent problemområde hvor det finnes noe tidligere forskning på hvilke verktøy som eksisterer. Likevel er det gjort lite kvalitativ forskning på hvordan slike verktøy benyttes i praksis. Derfor vil denne oppgaven ta utgangspunkt i hva man vet om tradisjonelle og smidige verktøy for å sammenligne erfaringer i arbeidslivet. Kapittel 5 – Datainnsamling - beskriver utarbeidelsen av en slik intervjuguide. Det er derfor hensiktsmessig at intervjuguiden tar for seg blant annet hvem som er med i valget av smidige verktøy, og på hvilke grunnlag de velges, i tillegg til hvordan utviklerverktøy, som for eksempel om testverktøy kan integreres med prosjektstyringsverktøy.

Valg av metode

Fremgangsmåte for å svare på problemstillingssettet er å samle data gjennom en ustrukturert intervjuform, transkribere datamaterialet og analysere ved hjelp av retningslinjer gitt av mal-analyse. Intervjuguiden utarbeides på bakgrunn av smidige prinsipper og kategoriseringer av verktøy beskrevet i teorikapittelet.

Potensielle informanter

Potensielle informanter er personer som har praktisk erfaring med smidige metoder. Problemstillingen tar ikke høyde for en sammenligning mellom smidige metoder og tradisjonelle metoder, og det er derfor ikke ønskelig å intervjue informanter som ikke har en smidig arbeidserfaring.

4.2 Fremgangsmåte for innsamling av data, analyse og diskusjon

På bakgrunn av valgte metoder presenteres følgende fremgangsmåte for å svare på problemstillingen:

1) Forberede intervjuguide med hensyn på tema som har fremkommet fra teorikapittelet, sett i forhold til problemstillingssettet, med hovedfokus på forholdet mellom bruken av verktøy og anvendt smidig metode i prosjekter.

2) Invitere potensielle informanter til intervju og gjennomføre intervju.

3) Innsamlet materialet kodes som beskrevet i metodekapittelet.

4) Mal utarbeides som definerer videre bearbeidelse med analyse. For å konsentrere innsamlet materialet om problemstillingssettet velges kategorier fra endelig mal som hovedfokus for analyse og diskusjon.

5 Datainnsamling

Sett i forhold til problemstillingen, valget av kvalitativ metode og mal-analyse, ble det neste steget å finne informanter som kunne fortelle om sine erfaringer med smidige metoder. Det var derfor nærliggende å invitere potensielle informanter med ulike prosjektroller. Det ble ikke lagt vekt på hvilke smidige metoder de respektive informantene har benyttet, men det var sentralt at de hadde erfaring med smidige metoder. En invitasjon til intervju ble sendt til 10 potensielle informanter hvorav 5 av disse responderte positivt og sa seg villig til å bli intervjuet. Tilgjengelighet var derfor avgjørende for hvilke informanter som stilte.

5.1 Intervjuguide

Som nevnt i metodekapittelet bør intervjuguiden knyttes direkte mot problemstillingen, i tillegg til forskerens egne erfaringer med forskningstemaet. En valid og pålitelig gjennomgang av valgte tema og forskerens egne meninger bør diskuteres med andre personer som har kjennskap til forskningsfeltet.

I lys av disse faktorene vil det være hensiktsmessig til hvilke verktøy som brukes, hvem de er valgt av og for hvilke problemområder de er ment å løse. For å utarbeide intervjuguiden ble X. Wang et al. (2010) benyttet for å utdype utfordringer i en smidig planleggingsfase. Denne studien tar for seg smidig utvikling for både samlokalisert og distribuert utviklingslag, og det er derfor viktig å inkludere krav for distribuerte verktøy.

Problemstillingen er formulert med særlig hensyn på prosjektstyringsverktøy og utviklerverktøy for å kunne gjennomføre smidig av prosjekter. Prosjektstyringsverktøy har i noen tilfeller integrerte moduler. For eksempel har JIRA moduler som kan installeres på plattformen, eksempelvis Confluence som fungerer som en wiki for organisasjoner og prosjekter. Det å få ett innblikk i informantens daglige bruk av prosjektstyringsverktøy og utviklerverktøy gjør at man kan kartlegge hvilke verktøy som er effektiv i ulike typer prosjekter, sett fra informantens rolle i prosjektet.

Intervjuguiden ligger i sin helhet i vedlegg 1. Først ble det stilt spørsmål til informanten om personens bakgrunn og erfaring med systemutvikling og smidige metoder. Deretter ble det stilt spørsmål omkring et fritt valgt smidig prosjekt slik at samtalen kunne kontekstualiseres. Avslutningsvis ble det stilt noen spørsmål omkring informantens opplevelser av ulike situasjoner ved bruk av verktøy. Disse spørsmålene kunne endres noe underveis fra informant til informant, etter erfaring fra intervju til intervju. Tekst markert i rødt for kategorien «Verktøy og opplæring» ble markert for å vise at disse kategoriene på en mindre direkte måte var knyttet opp mot kategorier som omhandlet smidige metoder. De andre kategoriene ble etter beste evne knyttet opp mot

elementer fra smidig metodologi.

5.1.1 Metoder representert av et tema

For å kvalitetssikre intervjuguiden ble temaceller merket med hvilke smidige metoder som var aktuell for den respektive cellen og hvilke smidige prosesser eller begrep den refererte til. Hvis cellen ikke hadde noe tilknytting til en smidig metode ble den likevel med fordi disse cellene var tema som var i direkte tilknytting til prosjektstyringsverktøy eller utviklerverktøy og som var essensielle med tanke på problemstillingen.

5.1.2 Egne innspill

Egne erfaringer med smidige metoder, som Scrum, XP, lean og kanban dannet begynnelse på utarbeidelsen av intervjuguiden. I tillegg til teori om smidige metoder og prosjektstyringsverktøy og utviklerverktøy la dette grunnlaget for intervjuguiden.

5.1.3 Prosjektstyrings- og programkodenivå

Arbeidet med å samle data fra informanter er særdeles viktig for at datasettet skal underbygge problemstillingen. Rekkefølgen man representerer de forskjellige temaene for den aktuelle informanten kan være avgjørende for å diskutere relevant informasjon først. Tabell 1 forsøker å belyse denne fremgangsmåten, hvor man kan bevege seg fra venstre til høyre hvis informanten har en prosjektstyringsrolle i det aktuelle prosjektet, og da følgende fra høyre til venstre hvis informanten var i en utviklingsrolle. Fra et smidig ståsted kan dette beskrives som verktøy for oversikt over prosjektet som helhet (prosjektstyringsnivå), for eksempel for en prosjekteier. På den venstre siden har en tema som er aktuelle på brukerhistorie- og oppgavenivå (programkodenivå), for eksempel for utviklere. Informanter har gjerne noe forståelse som overlapper disse, så alle celler ble forsøkt dekket i alle intervjuene. Den ønskede effekten av en slik oppdeling av intervjuguiden var mindre vellykket enn på forhånd antatt, som kan begrunnes med at informantene hadde god dekning på de fleste områder til tross for at de hadde en prosjekstyringsrolle eller utviklerrolle.

Prosjektstyringsnivå	**Programkodenivå**
Overlappende	

Tabell 1- Nivåkategorisering av intervjuguiden.

5.1.4 Informantens erfaring og prosjektvalg

Som en introduksjon til intervjuene var det interessant å bli kjent med informanten gjennom personens erfaring med smidige metoder i arbeidssammenheng. For å oppnå dette var det satt opp forhåndsvalgte spørsmål før selve temaene ble diskutert. Det var også ønskelig å holde seg til et «hovedprosjekt» som informanten hadde deltatt i og som var avsluttet. Det var nødvendig å få vite om antall personer i utviklingslaget og faktisk tidsbruk for prosjektet, slik at smidige metoder kunne diskuteres opp mot disse nøkkeltallene.

5.1.5 Roller og tverrfaglighet

Selv om viktighet av fremgangsmåte av intervjuet skal prøve å samle informasjon av informanter i best mulig rekkefølge, vil man møte personer som har høy tverrfaglig kunnskap innad i prosjektet og organisasjonen. Slike informanter kan gi gode data på hvordan ulike verktøy påvirker alle prosjektdeltakere, fra prosjektstyring til integrasjon.

5.1.6 Kommunikasjonsflyt og dynamikk

Er det mulig å tegne seg et bilde av hvordan informasjonsflyten fungerer i de ulike prosjektene informantene representerer? Kan dette kartlegges videre av ulike hjelpemidler som flytdiagram eller en kodetabell? Hvis man kommer over muligheten til slik data, kan man muligens kartlegge brudd i arbeidsflyt (innad i prosjektet) som en følge av feilvurdert verktøy?

For eksempel ved user case «programmerer trenger informasjon om templates». Videre kan det iht. mal-analyse være mulighet for å utheve tilfeller hvorpå mengde av indirekte arbeid kan være en kritisk faktor for om det smidige verktøy virker etter sin hensikt?

5.1.7 Kostnad og opplæring

Forskjellige kostnader som er relatert til hvor mye det koster prosjekt å ta i bruk et nytt verktøy. Dette blir gjerne evaluert i begynnelsen av et hvert prosjekt. Opplæring i seg selv er en kostnad som kan vise seg å være vel investert underveis i prosjektets levetid.

5.1.8 Test av intervjuguide

I tillegg til problemstillingen var grunnlaget til forberedelsen av intervjuguide mine egne erfaringer innenfor smidige metoder og praktisk bruk av prosjektstyringsverktøy i arbeidssammenheng og fagrelaterte prosjekter. Utarbeidelsen av intervjuguiden var en iterativ prosess hvor flere utkast ble

laget for å til slutt komme frem til en endelig guide. Intervjuguiden var videre basert på forskningsartikler relatert til krav for smidige verktøy i forbindelse med samarbeid, koordinasjon og gjennomføring av smidige prosjekt. Ustrukturerte samtaler med personer med kjennskap til forskningsfeltet var derfor viktig. I hovedsak var dette samtaler med veileder og medstudenter. Deretter gjennomførte jeg et testintervju av en medstudent hvor vi hadde en samtale om et prosjekt vi begge var deltakende i.

5.2 Gjennomføring av intervju

Intervjuene ble gjennomført i perioden desember 2011 til februar 2012. Det ble gjort totalt 5 intervjuer hvor intervjutiden lå mellom en halvtime og en time. To av intervjuene ble holdt i møterom på universitetet og tre intervjuer på arbeidsstedet til informanter. For å transkribere intervjuene ble en versjon av transkriberingsprogrammet Express Scribe brukt. Ingen spesielle metoder ble benyttet for å transkribere, men metainformasjon ble lagt til i tillegg til ren tale-til-tekst, for eksempel ved forstyrrelser, pauser eller i tilfeller hvor informanten virket å uttale setninger på en ironisk måte.

5.2.1 Koding av transkripsjoner

Etter transkriberingsprosessen ble hvert intervju gjennomgått for å gi et overblikk over datamaterialet. Intervjuene ble så kodet i programvaren R – en type gratisversjon av CAQDAS[40]. Modulen i R som ble brukt, RQDA, gjorde det mulig å opprette et prosjekt med filer som inneholdt koder, og kategorier av koder. Dette lettet noe av arbeidet med kodingen. Likevel var det mye arbeid med å kode hver enkelt transkripsjon i tråd med King sin beskrivelsen av mal-analyse. For eksempel var det ikke mulig å opprette nummerering på kodene. En tyngre versjon av CAQDAS hadde nok løst dette, men RQDA var nyttig for å kartlegge hvor i transkripsjonen ulike tema forekom. Det ble derfor ikke lagt så stor vekt på selve kodingen i dette programmet. Viktigere var det å få merket interessante utdrag slik at disse kom tydeligere frem ved neste gjennomgang av intervjuene. Etter gjennomgangen av hvert enkelt intervju ble et kortfattet sammendrag laget, slik at disse kunne fungere som referanser gjennom analysen.

5.3 Fra transkripsjon til mal-analyse

[40]Computer Assisted Qualitative Data Analysis Software

Ut ifra intervjuguiden[41] benytter jeg kategoriseringen, de overordnede temaene på venstre side, for å kode materialet. Disse kategoriene er listet opp i tabellen under. I tillegg hadde jeg, etter å ha transkribert materialet, blitt bevisstgjort over kategorier jeg kunne tilføye allerede før jeg gjorde selve kodingen på materialet.

A priori koder
Planlegging
Tverrfaglighet
Kommunikasjon og dynamikk
Samlokalisert utvikling
Visualisering
Verktøy og opplæring

Tabell 5: A priori koder

5.3.1 Første mal

I kodefasen for å utarbeide den første malen var det nødvendig å dekke hovedkategorier som var relevante for problemstillingen. Selv om for eksempel en kategori forekom i kun ett av intervjuene kunne likevel denne kategorien ha stor signifikans alene. Jeg gikk i hovedsak frem ved å se på det første transkriptet for å utarbeide en første mal, men jeg fortsatte ved å fylle ut malen ved å se på kategorier som fremkom av også i transkript nummer to og tre.

5.3.2 Utvikling av mal

Etter den første malen var utarbeidet på grunnlag av det første intervjuet, ble denne brukt på de resterende intervjuene. Der hvor jeg så at det var nye, relevante kategorier, plasserte jeg disse under eksisterende kategorier, eller opprettet nye. Denne tidkrevende prosessen ble gjennomgått med alle intervjuene. Denne nøye gjennomgangen av relativt få intervju var med på styrke det metodologiske forholdet til datamaterialet. Da jeg var ferdig med å anvende de inkrementelle malene på hele datasettet, sto jeg igjen med en endelig utviklet mal som tok for seg aktuelle kategorier og tema for alle intervjuene som senere skulle benyttes til analyse og diskusjon. I etterkant, for å konkretisere analysen i forhold til problemstillingen, kryssrefererte jeg problemstillingen med den endelige malen.

[41] Se vedlegg.

5.3.3 Endelig mal

Avslutningsvis i datainnsamlingen utviklet jeg en endelig mal til bruk for videre analyse og diskusjon. Malen blir presentert i Tabell 6 og viser koder som er funnet ved bruk av mal-analyse. De vil derfor ikke nødvendigvis følge kategorier som er diskutert, fordi de er utledet av innholdet i transkripsjonene, og som nevnt ovenfor ikke enda kryssreferert med problemstillingen.

1 Aktuelt prosjekt	1.1 Ressurser	1.1.1 Tid 1.1.2 Antall utviklere, testere, lag.
	1.2 Type prosjekt	1.2.1 Kontrakt 1.2.2 Distribuert eller samlokalisert
	1.3 Hierarki	1.3.1 Flatt eller flere grader
2 Smidige metoder	2.1 Planlegging/estimering	2.1.1 Planning Poker 2.1.2 Ingen estimering
	2.2 Utviklingsdrevet type – testing	2.2.1 Test-Driven Development (TDD) 2.2.2 Behaviour-Driven Development (BDD)
	2.3 Måling av hastighet/progresjon	2.3.1 «Velocity»
	2.4 Kommunikasjon og møter	2.4.1 Planlegging – Scrum: «Sprint planning» 2.4.2 Daglig møte – Scrum: «Daily standup» 2.4.3 Retrospektivmøte – Scrum 2.4.5 Demo – Scrum: «Sprint review meeting»
3 Prosjektstyringsverktøy	3.1 Visualisering	3.1.1 Fysisk plassering ifht. prosjektteamet. 3.1.2 Visualisering av oppsplitting av brukerhistorier 3.1.3 Fargebruk
	3.2 Implementasjon av PSV tilpasset metode	3.2.1 Grad av tilpassingsmuligheter/fleksibilitet
	3.3 Visualisering	3.3.1 Bruk av farger 3.3.2 Måling og synlighet av hastighet/progresjon i prosjektet
	3.4 Digitale verktøy	3.4.1 Prosjektendringsinformasjon 3.4.2 Bruk av verktøy i ulike faser i prosjektet
	3.5 Fysiske artefakter	3.5.1 Scrum-tavle 3.5.2 Lean / Kanban -tavle 3.5.3 Digital vegg
	3.6 Informasjonsnivå	3.6.1 Prosjektnivå 3.6.2 Iterasjonsnivå 3.6.1 Oppgavenivå
	3.7 Roller og og valg av verktøy	3.7.1 Kunderolle/kundeinnsikt 3.7.2 Pålagt bruk 3.7.3 Fritt valgt 3.7.4 Ulik oversikt for ulike roller
	3.8 Brukerhistorie -tilordning	3.8.1 Sporing av brukerhistorier fra splitting til oppgaver gjennom utviklingsfasene.
4 Konfigurasjons -og prosess-automatisete verktøy	4.1 Integrasjon med PSV	4.1.1 Tett integrert mot PSV 4.1.2 I liten grad integrert mot PSV
	4.2 Testverktøy	4.2.1 Testkrav
	4.3 Kontinuerlig integrasjon og	

	byggverktøy	
	4.4 Versjonskontrollsystem	4.1.1 Integrasjon med PSV

Tabell 6: Endelig mal – tallene representerer det vertikale kodehierarkiet med toppnoden ytterst til venstre slik at man kunne forholde seg til tall i stedet for lengre setninger under kodefasen.

Kryssreferanse med problemstillingen

Hver kategori i den endelige malen ble sammenlignet med problemstillingsettet ved å kryssnummerere hver enkelt problemstilling for 2.kategorinivå fra mal-hierarkiet i den endelige malen. For eksempel, når jeg hadde følgende mal-hierarki: 3 - «Prosjektstyringsverktøy»; 3.4: «Digitale verktøy»; 3.4.2: «Bruk av verktøy i ulike faser i prosjektet» nummererte jeg 3.4 med problemstilling nummer 1,3 og 4.

Valgte kategorier bør få mest oppmerksomhet i analyse og diskusjon. Andrenivåskategorier som ikke får en problemstillingsnummerering kan brukes hvor det er nødvendig for å kontekstualisere andrenivåkategoriene, som for eksempel tema vedrørende smidige metodologi.

5.3.4 Bruk av endelig mal på intervjuene

Malen ovenfor var mitt utgangspunkt i utvelgelsen av relevante sitat og eksempler fra datamaterialet. For å unngå å kun oppsummere innholdet i intervjuene, var det viktig å ha en egen diskusjonsdel for å knytte sammen eksemplene med teorien. Det er i stor grad eksemplenes forhold til teorien som former grunnlaget for konklusjonen og videre teoriutvikling.

5.3.5 Vurdering av reliabilitet, validitet og generaliserbarhet

Forskerrollen definerer parametrene rundt forskningssituasjonen, men det er vanskelig å garantere ekstern reliabilitet siden min sosiale rolle i intervjuene var unik. Siden denne studien tar for seg et lite utvalg, altså fem intervjuer av medarbeidere i systemutviklingsbedrifter i Bergen, kan den sies å en lav grad av ekstern validitet. Begrensede ressurser bidro til at det ikke var observatører til stede under intervjuene. En samarbeidsprosess med veileder angående intervjuguide, samt testintervju av en medstudent var med på å kvalitetssikre denne prosessen.

Studien kan likevel til en viss grad reproduseres ved å følge prosessen beskrevet i metode, forskningsdesign og datainnsamling, samt ved bruk av den vedlagte intervjuguiden. Mal-analyse belager på en subjektiv analytisk prosess, dette i likhet med andre kvalitative analysemetoder.

Personlige erfaringer fra systemutviklingsmiljø er med på å legge føringer for mine syn på smidige metoder. Dette gjorde at jeg som forsker i større grad hadde forståelse for situasjonene informantene fortalte om. Siden mine erfaringer er fra andre organisasjoner enn informantenes, var dette i tillegg til analytisk metodologi (mal-analyse) i sammenheng med det teoretiske bakteppet også med på å sikre et blikk utenfra.

Siden denne studien baserer seg på kvalitative data, er sitatene fra intervjuene ment for å være eksempler som skal sammenlignes og settes i en teoretisk kontekst. Det er derfor mer aktuelt i denne studien å se på forholdet mellom teori og eksempler som kommer frem i studien, enn å generalisere ut over datamaterialet..

6 Analyse

Selve analysen vil ta for seg informantenes erfaringer og bruk av verktøy og smidige metoder i systemutvikling. Dette for å gi et empirisk grunnlag for å drøfte problemstillingen i forhold til grounded theory og mal-analyse. Denne delen vil la informantenes syn komme frem, for å være åpen for hvordan empirien skal legge grunnlag for teoriutvikling, jamfør grounded theory.

Analysen vil gi svar på de ulike underproblemstillingene – altså hvilke verktøy informantene har tatt i bruk, hvilke utviklerverktøy de mener det er nødvendig å ha i smidige prosjekter, hvordan de opplever at funksjonaliteten støtter prosjektstyringsprinsippene for den aktuelle smidige metoden, og hvordan de opplever forholdet mellom datastøttede verktøy og artefakter.

For å gjøre det mer praktisk å analysere informasjon fra hver enkelt informant, presenteres alle informantene med bakgrunnsinformasjon, informasjon om det aktuelle prosjektet og hvilke verktøy som ble brukt. Prosjekter for hver enkelt informant blir diskutert og eksemplifisert. Dette er gjort for å få en oversikt over informantens erfaringer. Forskjellene i prosjektene gjør sammenligninger mellom ulike prosjekter og ulike informanter vanskelig, men det viktigste var å få varierte eksempler og et inntrykk av tendenser.

Informant	**Prosjektrolle**	**Digital prosjektstyring**	**Bruk av tavle**	**Metode**	**Prosjektteam**
Therese	Utvikler Scrummaster	JIRA	Scrum-tavle Kanban-tavle	Scrum/XP/Kanban	4 utviklere, en tester, hovedsaklig samlokalisert
Karl	Utvikler Teknisk leder	Team Foundation Server	Scrum-tavle Kanban-tavle (u/CONWIP);	Scrum/Kanban	6 personer, ett lag; Hovedsaklig samlokalisert
Vidar	Arkitekt	JIRA Team Foundation Server	Ingen	Iterativ Tett kommunikasjon	6 til 12 personer; Hovedsaklig samlokalisert
Stian	Scrummaster Teknisk leder	Scrumworks Team Foundation Server	Ingen	Scrum	6 til 18 personer; Noe distribuert
Fredrik	Teknisk leder	JIRA	Scrum-tavle	Scrum	13 personer m/ulik oppdeling av team.

6.1 Therese, utvikler

Therese jobber for en medium stor bedrift i Bergen som også har kontorer i flere byer i Norge. Hun er Scrummaster-sertifisert og har ellers en god del erfaring med smidig utvikling fra arbeidslivet og gjennom andre engasjement i 3-4 år.

Aktuelt prosjekt

I det aktuelle prosjektet som ble valgt for intervjuet var de omkring 3-4 utviklere, en tester og en prosjektleder, hvorav Therese var utvikler. I prosjektet skulle de utvikle et intranettsystem (nettløsning) for forsikringsbransjen. En tidligere versjon av systemet ble benyttet som et utgangspunkt og skulle ikke endres visuelt. Laget utviklet på timebasis og ikke på fast kontrakt, noe som gjorde at de brukte lite tid på estimering av oppgavene. En total oversikt over alle kravspesifikasjonen fra kunde ble formidlet til utviklerne sent i utviklingsprosessen. Dette, i tillegg til liten domeneforståelse av utviklerne, var en faktor for feilestimering og noe forsinkelse. Prosjektet holdt frem i ca ni måneder, men siden de arbeidet på timebasis var det på forhånd ikke definert en tidsramme. Prosjektet ble totalt sett beskrevet som suksess av prosjektteamet og kunde.

Smidige metoder

I den grad det var utnevnt en Scrum-master, fylte Therese den rollen: «Folk i teamet var ganske oppegående på smidig da, alle sammen», og hun antyder at de dermed ikke hadde et behov for en Scrum-master. Prosjektet var delt mellom flere avdelinger i Norge, men utviklingslaget var et selvstendig Scrumteam. De andre utviklingslagene benyttet seg ikke av Scrum. Teamet arbeidet noe distribuert, da noen av utviklerne jobbet fra hjemmekontor. Planlegging ble gjort for å estimere hvor mange timer en ville trenge for administrative oppgaver. Oppgavene ble estimert, men fordi det ikke var fastsatt eksplisitt dato for overlevering av produktet ble ikke dette vektlagt.

I et senere prosjekt ble planning poker brukt for å estimere hver iterasjon, og Therese beskriver estimering som overflødig og at det i tillegg er tidkrevende:

> «Jeg er egentlig ikke en stor fan av estimeringer. Så lenge ting må bli gjort, så må det bli gjort. (...) det blir aldri riktig».

Det ble praktisert mye ansikt-til-ansikt-kommunikasjon i teamet. Prosjektteamet fulgte Scrum ved å blant annet ha daglig standup ved tavlen i 15 minutter. Ett retrospektivmøter ble holdt, men kun ved prosjektets slutt. Therese beskriver ulike sider ved retrospektivmøtet «Det var jo sånn på en måte at dette skal vi ikke gjøre likt neste gang, sant, men vi gjorde jo det da.». Videre fortalte hun om mangel på oppfølging av ting som kunne bli gjort annerledes til neste prosjekt: «Dette var

dumt, men hvis du ikke har den andre delen som er, 'dette skal vi gjøre med det konkret', og faktisk følge opp at det blir gjort, så er det helt meningsløst».

Prosjektstyringsverktøy

Hovedsaklig ble en tilpasset Scrum-tavle brukt for å holde oversikt over pågående arbeid og progresjon. I tillegg til tavlen benyttet prosjektteamet seg av JIRA, som ble brukt innledningsvis i prosjektet for å holde orden på brukerhistoriene. Utover i prosjektet ble JIRA mindre brukt av utviklerne fordi de så behovet for å bruke tavle mer, og de brukte deretter JIRA mer til problemhåndtering. Derimot brukte testerne JIRA mer utover i prosjektet etter hvert som feil på produktet ble rapportert, i tillegg til Quality Center, altså en duplisering av verktøy som støttet problemhåndtering. Installasjonen av JIRA var valgt og satt opp av kunde: «[kunde] brukte det mye mer aktivt enn det vi gjorde». Therese oppsummerer av sin oppfatning av digitale prosjektstyringsverktøy med at «[jeg er] ikke veldig glad i sånne verktøy».

Fysiske artefakter

Therese nevner at det er «lettere å tilpasse en fysisk tavle til (...) min arbeidsprosess, skal jeg gjøre det i JIRA så er det et knot». For eksempel om det å legge til en ekstra kolonne for en ekstra fase i utviklingen nevner hun at «jeg kan tegne opp en ekstra kolonne på et whiteboard». Therese mener at ofte er det ønskelig å ha forskjellige oppdelinger av faser for ulike prosjekter, men hun synes det kan være tidkrevende å sette opp et egnet JIRA-oppsett for dette formålet. I tillegg hevder hun at en tavle er mer synlig fremfor en digital løsning: «[en tavle] er der hele tiden, den er synlig, alle kan se den. Ikke hvis man bare er utviklere, men også alle som går forbi».

I det aktuelle prosjektet brukte de Scrumtavle, men etter å ha erfart hvordan kanbantavle fungerte i senere prosjekter nevner hun hvordan brukerhistorier på en kanbantavle visualiserer flaskehalser på en bedre måte: «to av disse her venter egentlig på noen andre, (...) det trigger samtaler på en helt annen måte enn det en vanlig scrumboard.». Som en motsetning beskriver hun ironisk hvordan en Scrumtavle kan fungere: «nå har vi 15 ting 'in progress', men det er greit». Therese sier at de i noen tilfeller hadde bruk for å dele opp oppgaver/brukerhistorier når de kom inn i utviklingsfasen. For å synliggjøre oppdelingen av brukerhistorien markerte de oppgaver som tilhørte en brukerhistorie på kanbantavlen, slik at når alle underoppgavene var ferdige, så forentes de til den originale brukerhistorien som videre var klar til akseptansetestingsfasen.

Wiki

JIRA Confluence ble brukt som dokumentasjon for prosjektet. I hovedsak ble det brukt på slutten av prosjektet for å beskrive informasjon om systemet (produktet), slik at andre kunne få innsikt i

tekniske detaljer for videre vedlikehold.

Konfigurasjons- og prosessautomatiserte verktøy

I det aktuelle prosjektet brukte de Hudson som sendte ut e-post når integrasjonen og byggingen av programvaren feiler. Therese ser for seg bruk av Cucumber, et Behaviour-Driven Development - rammeverk for testing integrert med Hudson, i kommende prosjekter. Det var også en vurdering underveis å benytte Selenium for webtesting av klient-side, men tidspress satte en stopper for det.

6.2 Karl, utvikler

Karl er utvikler for et firma i Bergen som utvikler programvare for forsikringsbransjen. Han har 4-5 års erfaring som utvikler, og har i den senere tid fungert som teknisk leder.

Aktuelle prosjekter og smidige metoder

Gjennom intervjuet ble det valgt to prosjekter slik at man kunne se forskjeller mellom de. I det første prosjektet brukte de Scrum, hvor de skulle utvikle et helsesystem, hvor prosjektteamet hadde ansvaret for å lage en integrasjons-modul for et signatursystem. Prosjektteamet besto av fem utviklere og en tester. Prosjektet varte i seks måneder. Karl kom inn i prosjektet igjen når de hadde behov for tilpasninger et halvt år senere, i en periode på to måneder. Det var mye tidspress underveis i utviklingen, men prosjektteamet imøtekom estimatene litt over tiden. Prosjektet ble sett på som en suksess av kunde og Karl selv. Generelt sett sier Karl av erfaring at det går med mye tid med til Scrum-møter som planlegging og estimering, men at prosjektteamet ser positivt på et retrospektiv med demo-visning. Planning poker ble brukt for å estimere oppgaver, samt noe gjetting.

I det andre prosjektet begynte teamet med anvendelse av kanban. Utover i prosjektet, blant annet på grunn av tidsbegrensninger, tok de inn elementer fra Scrum, i form av en demonstrasjon av systemet for involverte aktører omtrent hver andre uke.

Karl sier at i begge prosjektene brukte de parprogrammering (i 95 prosent av tilfellene).

Prosjektstyringsverktøy

I begge prosjektene brukte de en digital løsning og tavle. Team Foundation Server var prosjektstyringsverktøyet for Scrum-prosjektet, hvor hovedsaklig estimater og faktisk brukt tid for oppgaver ble håndtert. Senere i prosjektet gikk de over til en kanban-tavle uten bruk av begrensninger av antall oppgaver i arbeid: «Vi hadde ikke noen sånn formalisering av (...) begrensinger på kolonnene for eksempel, vi hadde bare visualisering av hvor vi var i prosjektet».

Det forelå en form for mandat i bruk av Team Foundation Server, og Karl forklarer at personer i

teamet brukte Team Foundation Server akkurat så mye de måtte: «Hvis du endrer på noen tall inn i en eller annen forferdelig veiviser (...) i Team Foundation Server, så er det egentlig bare hassel, da gjør du akkurat så mye du må, og (...) hopper bestandig rundt systemet».

Karl forklarte at tidsbruk på å finne gode verktøy som ivaretar den valgte smidige metoden må balanseres:

> «Vi har vært veldig bra på å velge ny teknologi og forbedre oss selv, kanskje litt og,sånn at det har gått utover produktiviteten til tider (...). En håndverker, for å være effektiv så må han ta seg tid til å vedlikeholde verktøyene sine, men hvis han bare bruker tid å vedlikeholde verktøyene sine, så er ikke det en bra ting det heller»

I kanban-prosjekt sier Karl at de manglet en total oversikt over en slags brukerhistorieordning som kunne ha gitt bedre oversikt over hvordan brukerhistorier spores fra start til slutt gjennom utviklingsfasene, og hvilke oppgaver som er tilknyttet brukerhistoriene: «[jeg] savner for så vidt den totale oversikten over hvordan prosjektet som helhet går».

Teamlederen i prosjektet benytter seg av Excel for å holde oversikt over prosjektfremgang og estimater og progresjon av brukerhistorier gjennom utviklingsfasene. Det var ingen integrasjon mellom regnearket teamlederen hadde oversikt over, og det ble heller ikke kommunisert til utviklerne.

I kanban-prosjektet sier Karl at de brukte en kanbantavle i kombinasjon med Agile Zen:

> «[Vi bruker] et online kanbanboard kalt Agile Zen, i kombinasjon med fysisk kanbanboard. Personlig ville jeg helst sett at vi kun hadde ett fysisk et. Men vi har en del medlemmer av teamet som jobber en del hjemmefra. Da må du nesten ha en sånn elektronisk en i tillegg da, men å ha den fysiske, det, ja det er veldig mye mer synlig i et teamrom.»

Testeren bruker også Agile Zen, og flytter oppgaver fra testfasen til en godkjent-kolonne, eller hvis testen feiler, tilbake i sprintbakloggen. I begge tilfeller markerte de en farge på oppgaven, alt etter bestått eller feilet test.

Karl sier at de hele tiden forsøker å se på hvilke områder de kan forbedre seg. I forbindelse med bruk av prosjektstyringsverktøy var de fleksible og tok i bruk nye verktøy der hvor de så behov for det, også underveis i prosjekter.

Konfigurasjons– og prosessautomatiserte verktøy[42]

Karl sier at de har flere måter å prosessautomatisere oppgaver på, ved blant annet å bruke kontinuerlig integrasjonsverktøy. Versjonskontrollsystemet var ikke integrert mot

[42] Det ble generelt diskutert lite om akkurat hvilke typer utviklerverktøy de anvendte i Karls prosjekter, da intervjuet i stedet omhandlet en mer omfattende samtale om prosjektstyringsverktøy.

prosjektstyringsverktøyet, men hvem som helst hadde mulighet til å gå inn i kildekoden og se hvilke endringer som var gjort.

6.3 Vidar, prosjektarkitekt

Vidar arbeidet med drift av systemer i perioden 1998 til 2001, og har siden arbeidet med systemutvikling. Han har erfaring i roller som konsulent, seniorutvikler, sjefsutvikler og er per dags dato arkitekt. Vidar har ulike sertifiseringer i Java og rammeverk, men ingen innen metodologier.

Aktuelt prosjekt

Tidsperspektivet var noe flytende for prosjektet. Planleggingsfasen og utviklingen holdt frem i over ett år, men flere prosjektfaser bygger på komponenten som ble utviklet. Størrelsen på prosjektteamet var mellom to og fem utviklere. De skulle endre forretningslogikken for nettbanktjenester, basert på eldre programvare. Vidar anser prosjektet som en suksess og nevner en god balanse av forskjellige personer i teamet som en viktig faktor.

Smidige metoder

Generelt for smidige prosjekter sier Vidar at «vi tar ut elementer fra metodikken som vi tror passer oss best, også anvender vi de». Daglige møter ble ikke vektlagt da kommunikasjonen flyter bra i ett lite team. De benytter seg heller ikke av tavler og begrunner det med at «vi vet egentlig veldig mye hvem som jobber på hva til en hver tid». Valget om å arbeide på en smidig måte ble gjort i enighet med alle aktører fordi det var knyttet noe usikkerhet til bruk av ny teknologi og fordi halve prosjektteamet bestod av personer som tidligere ikke hadde jobbet sammen. På generell basis sier Vidar at halvparten av prosjektene han deltar i er basert på tradisjonelle metoder, mens den andre halvparten er mer smidig rettet prosjektgjennomføring.

Prosjektstyringsverktøy

Vidar sier at de brukte JIRA og GreenHopper for å håndtere oppgaver, men brukte ikke GreenHopper aktivt for en bestemt smidig metode. Vidar sier at de kommuniserte mye uten bruk av prosjektstyringsverktøy, men at det var mange samtaler innad i teamet. «Vi har (...) mer tradisjonelle styringsverktøy, og det er mer statisk. Men man løper jo mye rundt og snakker med folk og sjekker ut teknologier, rammeverk, måter å jobbe på, verktøy, ja kommuniserer veldig mye med andre». Vidar sier de ikke skrev på vegger fordi de visste til en hver tid hvem som jobbet på hva. Han understreker at for å få til en prosess er ikke nødvendigvis gitt ett verktøy: «Om det er mail eller du har workshops eller om det er å lage dokumenter eller wiki, så er det litt mindre viktig, men at vi kommer dertil er avgjørende».

Konfigurasjons- og prosessautomatiserte verktøy

Vidar sier at de benytter mange utviklerverktøy for å løse mange av utfordringene med konfigurering og prosessautomatisering, blant annet Jenkins[43] for kontinuerlig bygging av programvaren sammen med Maven. Atlassian Crucible og Atlassian FishEye blir brukt for å gjennomføre kildekodegjennomgang både av han selv og eksternt fra utlandet. Av testrammeverk sier Vidar at de blant annet bruker Mozilla, JUnit, Spring sin teststøtte og Apache Camel sin teststøtte. Han forklarer at testrapportering er fordelt på to verktøy: interne testrapporter, som for eksempel oppgaver, feil eller forbedringer opprettes i JIRA; eksterne feilrapporter fra kunde rapporteres i Quality Center.

> «Ideelt sett, så hadde de vært integrert, men det de vel ikke, så de får vi tilsendt på mail. Du kan delvis få de integrert i IDE'en din, avhenger av om du er på windows eller linux, men vi skulle aller helst hatt bi-direksjonal synching mellom JIRA og Quality Center, og tatt alt inn i IDE'en *fra* JIRA.»

Vidar nevner at det er tre lignende tilfeller hvor prosjektteamet også må forholde seg til Quality Center, og forklarer at det henger sammen med typisk verktøy for spesifikasjon på hvordan programvare skal testes. Vidar forklarer hvordan han som prosjektarkitekt har mulighet til å følge endringer i kildekoden ut ifra de retningslinjer som er beskrevet i «Software Architecture Document» (SAD).

> «[jeg får] en RSS-feed på alt som skjer sånn at i starten var jeg litt mer opptatt av å følge koden for å fange opp avvik fort da, og styre folk innpå hvordan det skal gjøres. Så etter en stund kan relaxe det litt, fordi folk har skjønt konseptene, så da er liksom informasjonsbehovet mer å se hva skjer, hvor er det aktivitet.»

Etter prosessen med oppfølging i begynnelsen av prosjektet, kan Vidar hente relevant informasjon fra rapporter i JIRA og Jenkins for testing, produktbygg, produktslipp og utrulling av programvaren. «Du kan egentlig følge den changen hele veien ut til produksjon», forteller han.

I det aktuelle prosjektet brukte de to ulike versjonskontrollsystem, både Subversion og Git. Han ville selv ha valg Git istedet for Subversion og begrunner det nærmere:

> «Cluet er at vi er pålagt visse rammer som kommer sentralt fra ut forbi ett prosjekt, og da heter det seg du skal bruke Subversion. Også vet jeg om jeg hadde kommet drassende med Git så hadde sikkert (...) mange ikke blitt overbevist, men nå kan vi kan på en måte være politisk korrekte. (...) <Vi> får veldig mange av fordelene uten å ofre å være politisk ukorrekt da, men jeg tror vi hadde hatt det enda bedre om vi hadde Git overalt.»

Å følge retningslinjer ved å bruke Subversion hemmer ikke prosjektteamet betraktelig mener Vidar, men han trekker likevel frem Git som fordelaktig over Subversion, som et fritt valgt

[43]Het tidligere Hudson. Det tidligere Hudson-miljøet anser Oracle Hudson som en gren (eng: fork) av gamle Hudson, mens Oracle Hudson anser Jenkins som en tilsvarende gren. http://en.wikipedia.org/wiki/Hudson_(software)#Hudson.E2.80.93Jenkins_Split

versjonskontrollsystem.

6.4 Stian, prosjektleder og Scrummaster

Stian har utviklererfaring igjennom 14 år og har brukt smidige metoder i fem år – hovedsaklig Scrum. Han var leder for det aktuelle prosjektet og han hadde også rollen som Scrummaster, som han tok sertifisering for i 2007. Han har jobbet som teknisk leder og arkitekt men har per dags dato roller som prosjektleder og kunderådgiver.

Aktuelt prosjekt

Firmaet selger tjenester i forbindelse med en plattform for finansiering og i dette prosjektet skulle de utvikle en nettløsning som bygger på plattformen. Prosjektteamet på seks utviklere var for det meste samlokalisert. Teamet jobbet selvstendig, men det var fleksibilitet mellom andre team på stedet ved behov. På generell basis må de forholde seg til at kunder er eid av selskaper i Europa og USA, slik at kontrakter blir estimert og spesifisert i forkant. Stian sier at «da møter du det første hinderet mot å kjøre en fullverdig Scrum, for da må alt spesifiseres up-front». Bedriften har en fossefallsprosess i forhold til kommunikasjon med kunde. Internt følger prosjektteamet Scrum-rammeverket.

Smidige metoder

Prosjektteamet besto av seks personer. I samme lokale arbeider to andre Scrumteam som jobber mot andre kunder og andre prosjekter, men på den samme plattformen. Det var følgende ikke interaksjon mellom teamene. Estimeringen for det aktuelle prosjektet ble gjort grundig, gjennom en analysefase opp mot kunde. Først når prosjektet var analysert, ble større oppgaver brutt ned til mindre oppgaver, og estimeres normalt av forretningsutviklere og utviklere fra prosjektteamet, som til slutt blir lagt inn i prosjektstyringsverktøyet Scrumworks. Tidligere hadde de spilt planning poker for å estimere oppgaver, men det ble ikke gjort for det aktuelle prosjektet. De som hadde best kjennskap til «området» estimerte, og diskusjoner kunne rundt størrelse på oppgaver oppstå hvis noen var uenige i estimatene. Prosjektteamet har gått vekk fra å ha daglig «standup», da de følte det var tid som kunne blitt gjort på andre arbeidsoppgaver. De fokuserer mer på statusmøter, og begrunner det med at det de har ellers tett kommunikasjon, slik at personer i teamet vet stort sett hvem som gjør hva.

Prosjektstyringsverktøy

Stian sier de bruker kontorprogrammer som regneark og tekstdokumenter for å gjøre grove estimeringer og dialog med kunde. På oppgavenivå blir estimater lagt inn i Scrumworks, hvor de blant annet har mulighet til å følge et nedbrennsdiagram, som hjelper de til å vite hvor mye teamet kan håndtere av arbeidsmengde i løpet av en uke eller en sprint. Kunde hadde ikke tilgang til innsikt

i Scrumworks, de får heller egne rapporter på nøkkeltall de er interesserte i. «[Kunde] har ikke tilgang til [ScrumWorks], mest fordi det gav de ikke de tallene de er ute etter, at de ville vite mer sånn om vi gjør jobben, når blir vi ferdig. Det var det som var viktig for de å vite».

Stian sier at «bugrapporteringssystemet»[44] deres er Team Foundation Server, men at de er på utkikk etter ett verktøy som integrerer dialogen for testing og supportfasen, i tillegg til å være personuavhengige og unngå kommunikasjon via e-post:

> ”Det vi ser på nå er jo å gå over til ett verktøy der du kan ha mer alt det samlet i ett, du har liksom dialogen mot kunde i prosjektet, og etter, i ett verktøy. (...) Vi har hatt andre som sliter litt med at det kanskje går enkeltpersoner, mailer frem og tilbake til kunde om bugs og alt sånt, at det ikke ligger i noen av systemene noen plasser. Så når den personen er vekk (...), så har vi ingen spor (...). Så det går mer på det at vi har all kommunikasjonen én plass og da kan du faktisk finne ut hva som har skjedd, uten at vi er personavhengige.”

Stian sier at kunde har tilgang til Team Foundation Server, og at kunde enkelt kan hente ut rapporter, men nevner at de ikke har en optimal løsning for å utveksle informasjon. I utgangspunktet har de en Sharepoint-løsning, men han forklarer at det å utveksle statiske dokumenter er noe tungvint, og at de heller skulle ønsket å ha en fungerende wiki-løsning: «Vi hadde en wiki-site oppe men, det var vanskelig å vedlikeholde, for det var typisk noe sånt som de som drifter systemene våre ikke syntes var gøy. De ville at det skulle være Microsoft».

Konfigurasjonsstyringsverktøy og prosessautomatiserte verktøy

Stian sier at de kjører enhetstesting og regresjonstesting, men at kunde selv står for brukertesting eller modultesting selv, som da blir et ansvar ovenfor kunde:

> «Vi kjører ikke den type brukertesting og modultesting og de tingene så mye her (...). I de aller fleste prosjektene så ønsker kunden selv <å teste>, da er de som er ansvarlige for testing, at de i hovedsak har interne som gjør det, de har sine verktøy som vi da kobler oss opp mot. Også har jo vi ett bugrapporteringssystem da som stort sett kunden rapporterer inn fra, fra sitt testsystem og inn i vårt».

(Stian hadde som tidligere beskrevet en prosjektlederrolle, dermed falt det naturlig å diskutere prosjektstyringsrelaterte emner i dette intervjuet, i tillegg til at prosjektteamet selv ikke sto for testrelatert arbeid).

6.5 Fredrik, Scrummaster

Fredrik har erfaring med smidige metoder siden 2005, men har også god kjennskap til blant annet eldre metoder som fossefallsmodellen, RUP og Method One. Han har hatt roller som ansvarlig leder, prosjektleder, Scrummaster og utvikler i ulike prosjekter. Han har Scrummaster-sertifisering

[44]Beskrevet tidligere som oppgavehåndteringsverktøy

og har arbeidet med opptil ti smidige prosjekter. Fredriks syn på utviklingsmetoder er å være bevisst på hva man skal levere og ut ifra gitt premisser, som ressurser og prosjekt og velge en passende metode.

Aktuelt prosjekt

Teamet skulle utvikle ny funksjonalitet på et eksisterende produkt, i forbindelse med rådgivning og beslutning for finansiering og forsikring. Prosjektbemanningen besto av tretten personer som utviklet over en periode på seks måneder. Oppdelingen av prosjektteam varierte fra sprint til sprint, da de i enkelte sprinter var et samlet prosjektteam, og i andre sprinter var de oppdelt i to til tre team. Samtlige var samlokaliserte, foruten en person i en annen by i Norge, som arbeidet med selvstendige oppgaver. I møter kommuniserte prosjektteamet med personen via telefon eller Skype.

Smidige metoder

I prosjektet fulgte de hovedsaklig Scrum. Prosjektteamet begynner prosjektet med å utarbeide en produktbaklogg som en intern oppnevnt kunde er med på å definere. De hadde standup hver dag, samt at de utøvde ukentlig sprintgjennomgang, for å koordinere prosjektteamene.

Prosjektstyringsverktøy

Fredrik har erfaring med flere prosjektstyringsverktøy, blant annet Assembla, Action Response, BugZilla og JIRA. Sistnevnte ble brukt i det aktuelle prosjektet. Fredrik sier at alle verktøyene gjør stort sett det samme, men det viktigste at prosessen bak oppgaven faktisk blir gjort. I et senere prosjekt hadde de valgt å ikke ta utgangspunkt i JIRA, men bare bruk av tavle. Utover i prosjektet valgte de likevel å ta i bruk JIRA igjen.

> «Det vi egentlig trenger er fire ting: hvem holder den [oppgaven]; hvor lang tid er det igjen; hva har du oppgave om å gjøre; og når er du ferdig. Og de fire tingene kan stå på en lapp. Og når du da må gjøre masse annet (...) beskrivelser av oppgaven som ikke gir noe innenfor de fire, så blir det ofte en hemsko»

Fredrik forklarer at det er kun et minimum av informasjon som trengs for å holde orden på oppgaver, og at prosjektstyringsverktøy med mye funksjonalitet kan være til hinder i form av tid for inntasting av mindre interessante detaljer om oppgaven. Det kan for eksempel være hvis en utvikler har eldre oppgaver fra tidligere prosjekter som fortsatt ligger i utviklerens baklogg. Fredrik beskriver fordelene ved bruken av fysisk tavle over JIRA:

> «Det går mye raskere, du får mye riktigere informasjon. Fordi at det når du flytter lappen, så er du der. Det er du som gjør det. Og det er mye enklere. I JIRA så har det en tendens til at du ikke får med deg alt som står på saken, sånn at du re-estimerer kanskje ikke, du glemmer å sjekke status på den, du sjekker ikke inn koden, altså det er en del ting som gjør at du kan være i inkonsistent tilstand, uten at noen vet det. Og det at det er faktisk den som eier saken som står og forteller og gjør transformeringen egentlig, (...) det gir en bedre effekt.»

Fredrik sier at dårlig vedlikehold av oppgaver i JIRA fører til inkonsistent informasjon, og at det kan forekomme uhåndterlige mengder av informasjon, som har kommet frem som et resultat av feil bruk av JIRA, som for eksempel manglende oppfølging ved omprioritering og opprydding i produktbaklogg. Prosjektteamet opprettet to prosjekter i JIRA; ett for planleggingsfasen, og ett for utviklingsfasen. De linket sammen de to prosjektene ved å lenke planleggingsoppgavene, samt forklaringer i Confluence, til utviklingsoppgavene. Confluence wiki fungerte ellers som dokumentasjon på tekniske detaljer. En annen side av prosjektstyringsverktøy mener Fredrik at «folk har en tendens til å distansere seg litt i verktøy».

Konfigurasjonsstyringsverktøy og prosessautomatiserte verktøy

I prosjektet ble Crucible og Bamboo brukt for å bygge programvaren. For de fleste utviklerne var Eclipse-installasjonen fri for innstikk. Prosjektteamet benyttet seg ikke av innstikk i Eclipse da utviklerne hadde oversikt over sine arbeidsoppgaver i JIRA på en separat skjerm.

7 Diskusjon

Malen var utgangspunktet for utvelgelsen av relevante sitat og eksempler fra datamaterialet. Ved å vise til eksempler forsøker jeg å belyse hvordan verktøy påvirker organisasjonens praksis av smidige metoder. Det er i stor grad eksemplenes forhold til teorien som former grunnlaget for konklusjonen og videre teoriutvikling.

Digitale prosjektstyringsverktøy og fysiske artefakter var en overordnet kategori fra den endelige malen og blir diskutert i egen seksjon, men disse er også relevante innenfor de undernevnte kategoriene. Drøftingen vil da ta for seg de kategoriene i den endelige malen som var mest relevant for problemstillingen: planlegging og estimering, roller og valg av verktøy, visualisering og synlighet, brukerhistorietilordning, versjonskontroll, samt kodegjennomgang. I disse kategoriene kom det frem eksempler i datamaterialet som belyste sider av det ofte uklare forholdet mellom prosjektstyrings- og utviklerverktøy, da jeg har fått inntrykk av at verktøy ofte integreres på ulike måter. Videre var eksemplene fra disse kategoriene med på å synliggjøre koblinger mellom verktøy, organisasjon og smidige metoder.

Noen tema blir ikke gitt like mye plass, og disse er plassert i forhold til diskusjonens kapittelseksjoner som etter beste evne beskriver temaene. Det vil også være noe overlapp i kapittelseksjonene da tema ofte er tett tilknyttet anvendelsen av smidige metoder og essensielle detaljer for tilfellene. Hver enkelt kategori forsøker å belyse påvirkningen av smidige metoder.

7.1 Planlegging og estimering

Digitale planleggings- og estimeringsverktøy som referert av X. Wang et al. (2010), for eksempel Rally, VersionOne, XPPlanner og Mingle ble ikke anvendt i de diskuterte prosjektene. ScrumWorks ble nevnt i ett tilfelle og Jira sammen med GreenHopper ble nevnt i tre tilfeller. Teamene var for det meste var samlokaliserte, slik at det var tilrettelagt for bruk av estimeringsmetoder som spilling av planning poker og estimering basert på tidligere prosjekter i stedet for digitale verktøy i planleggingsfasen av prosjektene.

I prosjektteamet til Stian var det to utviklere som jobbet utenfor Norge, men disse syntes ikke å være integrert i estimeringen av oppgaver. Disse to utviklerne var heller ikke integrert i andre faser av prosjektet, da de i stedet fikk rapporter i etterkant av møter. I dette tilfellet kunne et digitalt verktøy blitt anvendt for å integrere de eksterne utviklerne. Et spørsmål som kan reises i forbindelse med disse alternativene er om den ene måten er mer smidig enn den andre. Er det å integrere hele

prosjektteamet i planleggingsfasen, ved hjelp av et digitalt verktøy, en smidigere måte å arbeide sammen på sett i forhold til å fokusere på samlokalisert planlegging uten de eksterne utviklerne tilstede? Det kan muligens bestemmes ut i fra hvilke roller og hvilken erfaring utviklerne har, samt hvilke tilbakemeldinger de ville ha kunnet komme med tidligere i prosjektfasen. Likevel er det vanskelig å måle hvor stor påvirkning i anvendelse av smidig metode det ene eller det andre alternativet medfører.

De fleste informantene nevnte bruk av asynkrone prosjektstyringsverktøy med støtte for oppretting av brukerhistoriekort og oppgaver for planleggingsfasen. Eksplisitte synkrone planleggingsverktøy ble ikke nevnt, og ingen av informantene savnet en slik funksjonalitet, som videre kan tolkes at de samlokaliserte prosjektteamene er tilfreds med planning poker eller alternative metoder for smidig planlegging i form av tett kommunikasjon.

Digitale planleggingsverktøy kan hevdes å ha større verdi for distribuerte prosjektteam, jamfør X. Wang et al. (2010), og er begrunnet med at slike verktøy støtter opp om livssyklusen i smidige prosjekter. Scrum og XP vektlegger hvordan planlegging og estimering blir utført. For eksempel gir praksisen *The planning game* fra XP, samt kundes prioritering av produktbakloggen i Scrum, kontinuerlig fokus på verdi til kunde (Beck, 1999; Beck og Andres, 2004).

Prosjektteamet til Fredrik benyttet JIRA til holde på to prosjekter; ett for planleggingsfasen, og ett for utviklingsfasen. De lenket sammen de to prosjektene ved å merke lenker fra planleggingen til utviklingen i oppgavene, som viser at JIRA og GreenHopper er fleksible i måten å håndtere og lenke ulike faser i livssyklusen for prosjektet.

7.2 Visualisering og synlighet

Fredrik sa at «feilbruk» og et lite oppdatert prosjektstyringsverktøy (i dette tilfellet JIRA) resulterte i tidkrevende opprydningsarbeid. Han refererer til tendensen i det at utviklere glemmer å re-estimere, sjekke status, sjekke inn relevant kode for oppgaven og generelt sett at informasjonen er i inkonsistent tilstand uten at noen er klar over problemet. I videreført betydning innebærer dette at «definition of done» ikke overholdes. Dette eksemplet er etter min tolkning et av de viktigste argumentene mot bruken av digitale prosjektstyringsverktøy; det krever i bunn og grunn mer displin av prosjektteamet, imotsetning til fysiske artefakter man i utgangspunktet ikke kan unnlate å se, i tillegg til at fokuset rettes mot oppgavetavlen hver dag (eller gang) man har standupmøte.

7.3 Brukerhistorietilordning

Av analysen kom det frem eksempler på at brukerhistorietilordning i prosjektstyringsverktøy, der man har mulighet til å spore hvilken utviklingsoppgave som er tilknyttet en unik brukerhistorie. Therese nevnte at oppdeling av større brukerhistorier i utviklingsfasen, og at dette enkelt kunne synliggjøres på en kanbantavle. Man kan gjøre det samme i JIRA ved å lenke oppgaver seg imellom ved å merke et avhengigheter, men på et subjektivt grunnlag har jeg erfart at det må mange museklikk til, samt tilpassninger for å markere avhengighetsoppgaver. Therese pekte på dette som et av argumentene for hvorfor en fysisk tavle kan være mer fleksibel i det å tilpasse prosjektets arbeidsflyt imotsetning til for eksempel JIRA.

Eksemplene peker i retning av at fysiske artefakter er mer fleksibel og gir flere alternativer over digitale prosjektstyringsverktøy. En nærmere analyse av hvordan de ulike digitale verktøyene støtter en slik brukerhistorietilordning kan bidra til bedre hvilke tilpassninger som må til for at digitale verktøy skal kunne gi tilnærmet samme verdi som en fysisk artefakt.

7.4 Versjonskontroll

Anvendelse av versjonskontrollsystem som Subversion og Git ble nevnt av informantene, selv om det finnes flere varianter av både sentraliserte og distribuerte versjoneringsverktøy. Valg av blant annet versjonssytringskontroll blir nærmere diskutert i seksjon 7.6.

Jeg vil forsøke å belyse hvordan et versjonkontrollsystem kan fungere forskjellig i henhold til hvordan utviklingsoppgaver er delt opp (brukerhistorie, oppgave underliggende brukerhistorie, vedlikeholdsoppgave) og hvordan slike oppgaver er fordelt utviklere imellom. Et scenario som har kommet frem av datamaterialet, og spesielt i Thereses tilfelle, var å benytte en kanbantavle for vedlikeholdsprosjekter. Er det slik at Git eller Subversion også fremmer en arbeidsflyt som egner seg bedre til vedlikeholdsoppgaver eller mer omfattende nyutviklingsoppgaver?

Oppgavestørrelse og testing er faktorer som kan tale for eller imot distribuert eller sentraliserte versjonskontrollsystemer. Større oppgaver har kanskje generelt sett en mer kompleks integrasjon mot hovedgrenen og i slike tilfeller kan man kanskje dra fordel av å teste systemet lokalt (på utvikler-PC), altså på et distribuert versjonskontrollsystem. Mindre oppgaver, slik som vedlikeholdsoppgaver krever kanskje mindre avansert integrering mot hovedgrenen.

Jeg har gjennom analysenprosessen i denne studien blitt gjort oppmerksom på hvordan arbeidsflyten påvirkes av hvilket versjonstyringskontrollsystem som anvendes. Kan arbeidsflyten effektiviseres ved å dra fordel av de mulighetene distribuerte versjonskontrollsystemer gir? Alwis og Sillito (2009) viser til at for eksempel Git kan være mer egnet for distribuert utvikling. Da vil det være interessant å finne i hvilken grad Git er mer anvendelig for distribuerte smidige prosjektteam,

og hvordan Git påvirker den smidige livssyklusen? Samtidig kan det reises spørsmål om hvilke fordeler Git også kan ha for samlokaliserte team. Det at utviklere har full kontroll over kildekoden lokalt kan for eksempel korte ned tiden på tilbakemelding for regresjonstesting/enhetstesting ved å redusere ventetid av testresultat fra det sentraliserte byggverktøyet. En slik sparing av tid vil være i tråd med smidige prinsipper ved blant annet å minimere feil på hovedgrenen, "don't break the build" og bidra til en tettere forbindelse mellom utviklingskode og produksjonskode.

7.5 Prosjektstyringsverktøy og fysiske artefakter

Prosjektteamet til Stian brukte ScrumWorks for håndtering av oppgaver. Bare intern informasjon i teamet ble delt i ScrumWorks på brukerhistorienivå, og andre aktører hadde ikke tilgang til det. I prosjektteamet til Vidar var de ikke opptatt av å bruke tavler for å visualisere prosjektet, og støttet seg til tett kommunikasjon og det digitale prosjektstyringsverktøyet.

Prosjektteamet til Karl hadde duplisert den digitale løsningen med den fysiske tavlen i begge prosjekter. Årsaken kan være at størrelsen på prosjektteamet, fem utviklere i det første og ti utviklere i det andre, medførte at de måtte ha en måte å kommunisere med utviklere utenfra. Kompleksiteten for å utveksle informasjon øker når man har større prosjektteam, og kanskje spesielt når det er distribuert. Karl beskrev at en slik duplisering fungerte bra. AgileZen er et enkelt smidig verktøy som i hovedsak fokuserer på den smidige livssyklusen og faser i prosjektet. Fordelen med AgileZen var at det tok kort tid å vedlikeholde, samt at testere kunne gå inn og oppdatere, merke problemer, og generelt vedlikeholde prosjektstatus på en enkel måte.

Løsningen ivaretok smidige prinsipper i form av samtaler foran tavlen, i tillegg til at prosjektdeltakere og testere kunne følge progresjonen utenfra kontoret. Det kunne virke som de også dro nytte av at testere arbeidet mot den digitale kanbantavlen i AgileZen, imotsetning til tilfeller nevnt av de andre informantene. Det at de også på en enkel måte kunne endre arbeidsflyten (kolonner) i AgileZen gjorde at de til en hver tid hadde oppdatert informasjon, som senere ble «synkronisert» med den fysiske kanbantavlen. Dermed dro de også fordel av at brukerhistoriekort kan lagres og søkes opp i ettertid.

Therese beskriver JIRA som mindre nødvendig utover i prosjektperioden da de foretrakk å bruke en fysisk tavle. Samtidig påpeker hun at det kan være vanskelig å sette opp en skreddersydd JIRA + GreenHopper for flere prosjekter som kan ha ulik smidig livssyklus, som i følge Therese medfører en tidkrevende prosess for oppsettet. JIRA har støtte for smidig funksjonalitet, men kan også hevdes å ha for mange valgmuligheter som kan medføre et kompleks oppsett. Therese sa at det krevde en del ressurser for å tilpasse JIRA til egen arbeidsflyt, for eksempel med tanke på å opprette kolonner

for nye faser man kommer på i senere prosjekter.

Verken Vidar eller Stian sine prosjektteam anvendte fysiske artefakter for å synliggjøre pågående arbeid. I Stians tilfelle sa han at ScrumWorks fungerte for håndtering av brukerhistorier og estimater, og at den generelle kommunikasjonen i teamet var basert på mye ansikt-til-ansikt-kommunikasjon. Arbeidet med å estimere brukerhistoriene ble gjort av dedikerte utviklere. Min oppfatning av smidig estimering er at man skal kunne trekke inn synspunkter også fra de mindre erfarne utviklerne som dermed representerer helheten av prosjektteamet. Hvis en mindre erfaren utvikler utfører en oppgave som er estimert av en mer erfaren utvikler vil sannsynligvis oppgaven ta lengre tid enn hva som var forespeilet. Likevel ser jeg ut ifra informantenes synspunkter at estimering er mindre vektlagt enn hva smidige retningslinjer beskriver, enten det skyldes tidspress eller at utviklere ikke ser behovet for detaljert estimering.

Vidars prosjektteam anvendte JIRA og GreenHopper for håndtering av oppgaver, og begrunnet det med at i små team vet utviklerne hvem som gjør hva til en hver tid. Beck og Andres (2004, s.45) hevder de ikke har sett en fungerende løsning på å digitalisere brukerhistorier. Det er et skille mellom å vite hva personer jobber med, og hva de gjør til en hver tid, men som igjen avhenger av faktorer i prosjektet, for eksempel antall prosjektdeltakere. Et mindre team på 4-5 personer kan støtte seg til et digitalt prosjektstyringsverktøy og generelt tett kommunikasjon, men jeg vil likevel hevde at de dermed ikke drar fordel av formelle og uformelle samtaler rundt en fysisk artefakt. I tillegg vil man ikke få den effekten av synlighet fra informasjonsradiatoren tavlen kan gi for andre involverte aktører. En fordel ved bruk av digitale prosjektstyringsverktøy er å kunne ha informasjon om oppgaver integrert i IDE'en for å skaffe en enkel oversikt over arbeidsoppgaver. Blant annet har Visual Studio og Eclipse støtte for slike tilleggsfunksjoner, og informantene hadde ulike løsninger på hvordan de hentet informasjon fra det prosjektstyringsverktøyet; som nevnt kunne forefallende oppgaver hentes direkte fra IDE; rett fra prosjektstyringsverktøyet; direkte avlesning fra fysisk artefakt.

Anvendelse av kontorprogrammer

Prosjektteamet til Stian benyttet seg av kontorprogrammer for å skrive ned grove estimater, for deretter å legge inn estimeringer av brukerhistoriene på oppgavenivå i Scrumworks, i likhet med Vidar som refererte til bruk av regneark for å prioritere milepæler, tjenester og forslag. Karl nevnte at prosjektlederen førte estimater og budsjetter i regneark. Fellestrekkene kan synes å være at bruken av kontorprogrammer brukes for dokumenter som ikke har verdi gjennom hele

prosjektfasen, men heller i begynnelsen av prosjektet. Omfanget av anvendelse av kontorprogrammer samsvarer med Dubakov og Stevens (2008) og Azizyan et al. (2011). Fordelene kan synes å være at de egner seg ofte til grove estimater eller mindre avanserte beregninger for progresjon, samt at kontorprogrammer er lette å ta i bruk for de fleste aktører i smidige prosjekter.

Det som taler imot bruken av kontorprogrammer er i første omgang mindre synlighet. På et generelt grunnlag er det færre muligheter for å integrere relevant informasjon i dokumenter og regneark i andre verktøy.Sett opp mot prinsipper for smidige metoder vil ikke alle prosjektdeltakere ha tilgang slike filer, som i Karls tilfelle kunne ha dratt fordel av gjennomsiktighet av dokumenter mot prosjektteamet. Altså kommunikasjonen hindres av verktøyets begresninger i samarbeid. Likevel kan også kontorprogrammer hevdes å være effektive, de krever for eksempel lite opplæring og er lette å bruke. Hvis aktører behøver tilgang til informasjon i dokumenter og regneark, kan disse gjøres, er synlig ved å publisere dokumenter i relevante informasjonssystemer som for eksempel vedlegg i en wiki eller lignende plattformer.

7.6 Roller og valg av verktøy

Valg av verktøy kan ha innvirkning på faktorer som virker inn på hvordan smidige metoder blir anvendt i de respektive prosjektteamene.

Utveksling av statiske dokumenter gjør informasjonen vanskelig å håndtere. Det at en ikke har optimale løsninger for å utveksle informasjon blir en begrensning. Stians prosjektteam var begrenset til Sharepoint, som løste ikke de ønskede arbeidsoppgavene, da de hadde sett for seg bruk av en wiki-løsning. I dette eksemplet var det et valg som ikke ble bestemt av prosjektteamet, men som hadde konsekvenser for deres informasjonsutveksling. Interessekonflikten mellom driftteamet og prosjektteamet påvirket prosjektteamets mulighet til å legge premissene for deres egne informasjonsutveksling.

Vidar nevner at han ønsket å kun bruke Git fremfor Subversion og Git sammen. I dette tilfellet var det retningslinjer utenfor prosjektet som veide tyngst, og Vidar så ikke mulighet til å påvirke disse retningslinjene. Prosjektteamet møtte motstand i retningslinjer som er lagt utenfor prosjektet, men de kom seg rundt problemet på grunn av at Subversion og Git lot seg kombinere.

7.7 Kodegjennomgang

En felles forståelse av retningslinjer for hvordan en skriver kode og den interne læringsprosessen som følger er viktig for smidige metoder, som for eksempel oppnås via parprogrammering og

kollektivt eierskap i XP. Vidar nevner bruk av FishEye og Crucible ble brukt for å «outsource» teknisk kodegjennomgang til utenlandske oppdragstakere, i tillegg til at Vidar selv stod for kodegjennomgang i forbindelse med forretningslogikk, samt oppfølging av retningslinjer gitt i «software architechture document» (SAD). Verktøyene gir dermed mulighet for intern læring, men brukes i dette tilfelle også til outsourcing av teknisk kodegjennomgang. I lys av smidige metoder mener at jeg slike verktøy både støtter opp om smidige metoder, i form av umiddelbar tilbakemelding av skrevet kode, samt at verktøyene også kan hindre en intern læringsprosess. En intern utvikler kan få bedre forståelse av hvilke retningslinjer som ligger til grunn for det aktuelle prosjektet og i SAD'en ved å gå gjennom kode sammen med en mer erfaren utvikler. Verktøy som FishEye og Crucible kan forenkle denne prosessen. Samtidig øker slike verktøy også synligheten av innsjekket kildekode, for eksempel i nettleseren eller som Vidar nevnte å få RSS-strøm på hendelser man abonnerer på.

Oppsummering

Det er vanskelig, om ikke umulig, å bestemme i hvilken grad et prosjektteam er smidig. Er det slik at vi ser på prosjektteam som benytter fysiske artefakter som mer smidige enn de som anvender digitale prosjektstyringsverktøy? Vil en ytterligere utvikling av verktøy som samsvarer med den smidige livssyklusen gjøre at man går i retning av en mindre skarp motsetning mellom samspill og verktøy? Disse spørsmålene vil jeg ikke forsøke å svare på her, men fra et subjektivt ståsted og at jeg muligens er farget av de innvendinger informantene har representert gjennom denne studien mener jeg man må vurdere i hvert tilfelle hva som egner seg for hvert enkelt prosjekt.

Til slutt vil jeg vise til Vidars syn på verktøy, som gir mulighet til å zoome ut av verktøyboblen:

> «For å få prosessen til er ikke nødvendigvis et gitt verktøy, om hvordan det blir til, om det er mail eller du har workshops eller om det er å lage dokumenter eller wiki, så er det litt mindre viktig, men at vi kommer der til er avgjørende»

7.8 Evaluering av mal-analyse

Mal-analysen har i dette prosjektet fungert bra med hensyn på hvordan data ble innhentet og analysert. Jeg ble ikke overrasket i løpet av prosessen, siden jeg behandlet datamaterialet gjennom å kontinuerlig utarbeide det som til slutt skulle bli den endelige malen. Da jeg satt igjen med den endelige malen hadde jeg arbeidet tilstrekkelig med materialet til å vite hvordan analyse og

diskusjon ville bære frem. Jeg mener at man gjennom en slik analyse avdekker temaer som er essensielle uten å nødvendigvis måtte ha sammenlikninger mellom ulike informanters syn på verktøy og smidige metoder, og i stedet peker på enkelttifeller som like viktige. Malen hjelper til med å strukturere materialet slik at man avdekker funn som svarer på problemstillingen.

8 Konklusjon

I denne studien har jeg tatt utgangspunkt i manifestet for smidig programvareutikling, især personer og samspill fremfor prosesser og verktøy, og jeg har sett på motsetningen i økende anvendelse av verktøy i smidige organisasjoner og prosjektteam. Studien tok for seg hvordan smidige verktøy påvirker organisasjonens praksis av smidige metoder. Intervjuer og mal-analyse ble valgt som metoder for å utforske og analysere det smidige miljøet og hvordan systemutvikling fungerer i praksis. Metoden tok høyde for en åpen problemstilling ved å finne forhold mellom smidige metoder og anvendelse av verktøy i konsulentfirmaer i Bergen.

En tendens som kom frem av analysen var at prosjektteamene var opptatt av hvordan de selv kunne forbedre seg som team og utviklingsprosesser de la til grunn i utviklingsfasene, i tillegg til å endre prosjektstyringsverktøy og konfigurajons- og prosessautomatiserte verktøy i tråd med anvendt smidig metode. I flere tilfeller ble det nevnt at slike endringer også kunne forekomme underveis i prosjekter, for eksempel sjonglering mellom elementer fra Scrum og kanban. Dette gjorde det også vanskeligere å analysere bruken av smidige metoder mot bruken av verktøy. Som det også ble referert til i teorien, savnet informantene funksjonalitet for de aktuelle verktøyene, som for eksempel bedre mulighet til å visualisere hele prosjektforløpet, bedre rapporteringsmuligheter for ulike roller og generelt sett ha en mindre grad av duplisering av verktøy. Sistnevnte var ofte forårsaket av distribuert testing, av kunde eller andre parter, som anvendte egne verktøy.

I henhold til kontinuerlig forandringer i smidige metoder for de respektive informanters prosjektteam ble fleksibilitet i verktøy verdsatt av informantene slik at de kunne ha kontroll over teamets egenvalgte arbeidsprosess. Prosjektteamene hadde ulike måter å kombinere fysiske artefakter med digitale verktøy. Andre brukte enten fysiske artefakter eller digital prosjektstyring. Generelt for prosjektteamene var at de brukte Scrum som rammeverk, men at flere hadde dratt inn andre smidige eller lean elementer inn i rammeverket, som kanban eller utvalgte XP-praksiser. Informantene som anvendte kanban og lean utviklingsmetode sa at kanbantavlen egner seg til å se flaskehalser og øke synligheten av pågående arbeid, uavhengig om det var en fysisk eller digital tavle. Det kom også frem at tidsbegrensninger fra Scrum var motiverende for å visualisere progresjon i kanbanprosjekter, da Scrumtavlen og kanbantavlen viser ulik informasjon. En kanbantavle understreker hvor ressurser i et prosjektteam bør settes inn for å løse flaskehalser, mens en Scrumtavle sammen med et nedbrennsdiagram viser et mer helhetlig bilde av sprinten.

Selve valget av smidige metoder i prosjekter virket å være tatt av de respektive teamene. Lavt hierarki i organisasjonen kan være en årsak til dette, som la til rette for en intern «empirisk» anvendelse av smidige metoder. Selv om det forelå lite dirigering ovenfra-og-ned i organisasjonene fant jeg i enkeltetilfeller føringer gitt av organisasjonen som hindret fritt valg av verktøy.

Som et resultat av at digitale prosjektstyringsverktøy er mindre synlige enn fysiske artefakter kan det føre til mye informasjon i systemet som er irrelevant for informasjon om pågående arbeid, i tillegg til at avanserte verktøy tilbyr overflødig funksjonalitet, som taler mot anvendelse av digitale prosjektstyringsverktøy i smidige prosjekter.

8.1 Videre forskning

Jeg fant blant annet at det i dag eksisterer integrasjon av ulike verktøy som hjelper smidige prosjekter til å følge praksiser fra XP og arbeidsmetoder fra Scrum. Likevel har jeg ikke avdekket på hvilken måte disse integrerer, eller hvordan de ulike verktøyene har støtte for enkel integrering. Det kan være at verktøyet tilbyr et praktisk applikasjonsprogrammeringsgrensesnitt (API) for å lage egne moduler eller egne rapporteringsverktøy. For eksempel kan en kvantitativ studie, som viser hvilke verktøy som blir brukt i enkelte tilfeller, tolkes gjennom en påfølgende kasusstudie som avdekker på hvilken måte disse integreres. For eksempel kan dette kan gjøres ved å samle generert data av verktøy og avlesning av loggfiler for de aktuelle verktøyene.

Jeg fant det interessant at noen prosjektteam benyttet Git, mens andre holdt seg til Subversion. Selv om det hevdes at Git har bedre støtte for distribuerte team, kunne det være aktuelt å se på hvilke endringer i arbeidsflyt, og derav anvendelse av smidige metoder en slik overgang medfører også for samlokaliserte prosjektteam. Vil en tettere kobling mellom kildekode og arbeidsoppgaver gi en økt synlighet og gjennomsiktighet av pågående og gjort arbeid? I og med at det idag finnes webapplikasjoner, som for eksempel Github[45] med integrert problemsporingshåndteringmodul, kan en videre se på om det smidige miljøet, i form av kvalitativ forskning, går i denne retningen og i så fall hvordan utfallet blir med hensyn på synlighet og gjennomsiktighet av pågående og gjort arbeid.

[45] https://github.com/

Til sist vil jeg argumentere for fokus på flere kvalitative studier som på generell basis tar for seg prosjektstyringsverktøy, utviklerverktøy og anvendelse av disse innen smidige prosjekter. For meg bekjent er det få kvalitative studier som tar for seg denne problemstillingen, og slik litteraturen i denne studien viser vil den økende samlingen av prosjektstyringsverktøy og utviklerverktøy vi ser i dag åpne for diskusjon sett i forhold til det smidige manifestet for programvareutvikling.

9 Referanser

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods. *VTT publications, 478*(3), 3-107.

Amescua, A. B. n., L.; García, J.; Sánchez-Segura, M.-I. (2010). Knowledge repository to improve agile development processes learning. *Software, IET, 4*(6), 434-444. doi: 10.1049/iet-sen.2010.0067

Azizyan, G., Magarian, M. K., & Kajko-Matsson, M. (2011, 7-13 Aug. 2011). *Survey of Agile Tool Usage and Needs.* Paper presented at the AGILE Conference (AGILE), 2011.

Beck, K. (1999). *Extreme programming explained : embrace change*. Boston, Mass.: Addison-Wesley.

Beck, K., & Andres, C. (2004). *Extreme programming explained: embrace change*. Boston: Addison-Wesley.

Beck, K., Beedle, M., Bennekum, A. v., Cockburn, A., Cunningham, W., Fowler, M., . . . Thomas, D. (2012). Manifestet for smidig programvareutvikling. [Webside].

Boehm, B. W. (1988). A Spiral Model of Software-Development and Enhancement. *Computer, 21*(5), 61-72.

Bryman, A. (2008). *Social research methods*. Oxford :: Oxford University Press.

Cockburn, A. (2000). Characterizing people as non-linear, first-order components in software development.

Cockburn, A. (2003). *People and Methodologies in Software Development.* PhD thesis, University of Oslo, Norway.

Confluence, A. (2012). Confluence.

de Alwis, B., & Sillito, J. (2009, 17-17 May 2009). *Why are software projects moving from centralized to decentralized version control systems?* Paper presented at the Cooperative and Human Aspects on Software Engineering, 2009. CHASE '09. ICSE Workshop on.

Dencheva, S., Prause, C. R., & Prinz, W. (2011). Dynamic Self-moderation in a Corporate Wiki to Improve Participation and Contribution Quality

ECSCW 2011: Proceedings of the 12th European Conference on Computer Supported Cooperative Work, 24-28 September 2011, Aarhus Denmark. In S. Bødker, N. O. Bouvin, V. Wulf, L. Ciolfi & W. Lutters (Eds.), (pp. 1-20): Springer London.

Dubakov, M., & Stevens, P. (2008). Agile Tools. The Good, the Bad and the Ugly. [Whitepaper].

Dyba, T., & Dingsoyr, T. (2009). What Do We Know about Agile Software Development? *Software, IEEE, 26*(5), 6-9. doi: 10.1109/ms.2009.145

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology, 50*(9–10), 833-859. doi: 10.1016/j.infsof.2008.01.006

Goth, G. (2009). Agile Tool Market Growing with the Philosophy. *IEEE, 26*(2), 88-91.

GreenHopper, A. (2012). Atlassian GreenHopper.

Hunt, J. (2006). Tools to Help with Agile Development *Agile Software Construction* (pp. 217-237): Springer London.

JIRA, A. (2012). JIRA Retrieved 27-05, 2012, from http://www.atlassian.com/software/jira/overview/screenshot-tour

Kelter, U., Monecke, M., & Schild, M. (2003). Do we need 'Agile' software development tools? *Objects, Components, Architectures, Services, and Applications for a Networked World, 2591*, 412-430.

King, N. (2011, August 3). Template Analaysis, from http://www2.hud.ac.uk/hhs/research/template_analysis/

Kvale, S. (1997). *Det kvalitative forskningsintervju*. Oslo: Ad notam Gyldendal.

Larman, C., & Basili, V. R. (2003). Iterative and incremental developments. a brief history. *Computer, 36*(6), 47-56. doi: 10.1109/mc.2003.1204375

Molokken-Ostvold, K., & Haugen, N. C. (2007, 10-13 April 2007). *Combining Estimates with Planning Poker--An Empirical Study.* Paper presented at the Software Engineering Conference, 2007. ASWEC 2007. 18th Australian.

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*: Addison-Wesley.

Power, K. (2011). Using Silent Grouping to Size User Stories

Agile Processes in Software Engineering and Extreme Programming. In A. Sillitti, O. Hazzan, E. Bache & X. Albaladejo (Eds.), (Vol. 77, pp. 60-72): Springer Berlin Heidelberg.

Prause, C. R., Scholten, M., Zimmermann, A., Reiners, R., & Eisenhauer, M. (2008, 17-20 Aug. 2008). *Managing the Iterative Requirements Process in a Multi-national Project Using an Issue Tracker.* Paper presented at the Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on.

Royce, W. W. (1970). Managing the development of large software systems: Concepts and techniques. *In Proceedings of IEEE WESTCON. (Los Angeles, CA)*, 1-9.

Sarkan, H. M., Ahmad, T. P. S., & Bakar, A. A. (2011, 13-14 Dec. 2011). *Using JIRA and Redmine in requirement development for agile methodology.* Paper presented at the Software

Engineering (MySEC), 2011 5th Malaysian Conference in.

Schwaber, K. (2004). *Agile project management with Scrum*. Redmond, Wash.: Microsoft Press.

Schwaber, K., & Beedle, M. (2001). *Agile software development with Scrum*. Upper Saddle River, NJ: Prentice Hall.

Sharp, H., Robinson, H., & Petre, M. (2009). The role of physical artefacts in agile software development: Two complementary perspectives. *Interacting with Computers, 21*(1-2), 108-116.

Sigerud, K., & Baggiolini, V. (2011). The Software improvement process - Tools and rules to encourage quality. *Proceedings of ICALEPCS2011, Grenoble, France*.

Strauss, A. L., & Corbin, J. M. (1990). *Basics of qualitative research: grounded theory procedures and techniques*: Sage Publications.

Symon, G., & Cassell, C. (1998). *Qualitative methods and analysis in organizational research: A practical guide*: Thousand Oaks, CA: Sage Publications Ltd.

Wang, X., Conboy, K., & Cawley, O. (2012). "Leagile" software development: An experience report analysis of the application of lean approaches in agile software development. *Journal of Systems and Software, 85*(6), 1287-1299. doi: 10.1016/j.jss.2012.01.061

Wang, X., Maurer, F., Morgan, R., & Oliveira, J. (2010). Tools for Supporting Distributed Agile Project Planning. In D. Šmite, N. B. Moe & P. J. Ågerfalk (Eds.), *Agility Across Time and Space* (pp. 183-199): Springer Berlin Heidelberg.

10 Vedlegg

Vedlegg 1 – Intervjuguide

Innledning

Først ønsker jeg å spørre deg om din bakgrunn innen programutvikling og hvilke roller du har hatt?

- Har du sertifisering innen smidige metoder?
- Hvor mange år har du erfaring innen smidige metoder?
- Hvor mange prosjekter har du deltatt i?
- Har du erfart forskjellige metoder av smidig utvikling?
- Hvilken metode foretrekker du selv?

La oss gjerne ta utgangspunkt i det siste smidige prosjektet du deltok i, var det lenge siden?

- Grovt skisset, hva skulle dere utvikle?
- Hva var størrelsen på prosjektet med tanke på personer og antall team?
- Over hvor lang tid utviklet dere?
- Ble prosjektetresultatet mottatt som en suksess hos kunde? Hva var din personlige vurdering?

(Fortsetter på neste side)

TEMA	**UTVIKLERE, DESIGNERE**	**METODER**	**PROSJEKTEIER, SCRUMMASTER, PROSJEKTLEDER**	**METODER**
Planlegging [Husk erfaring]	Estimering, Iterasjon vs scopet	[*] Planning poker [*] Andre est. måter.	Kommunikasjon med kunde	[X,S] Kunde på stedet
	Iterasjon/slipp	[*] Små produksjons-slipper	Velositet - måling av produktivitet.	[*] Planlegging av ny sprint, nytt prosjekt
	Brukerhistorier	[*] Epic, User Stories, Tasks	Flytting av Stories fra iterasjon/release	[S] Sprint
Tverrfaglighet [Husk erfaring]	Oppdeling av roller	[*] Laginndeling: Scrummaster, utviklere, grafikere, kunde, produkteier, testere	Autonomitet	[*] Selvstyrt lag
	Roller	[*] Skjerme utviklere for ekstern støy	Kundeinvolvering og integrasjon i verktøy.	[*] Enhetsstesting, Brukertesting, Akseptansetesting/ Kundetest Kvalitetssikring internt.
Kommunikasjon og dynamikk [25:00] [Husk erfaring]	Endringer i prosjektet [*] Docs [*] Excel	[*] Bevisstgjøring av endringer	Brukerendringsstrøm	[*] Synlighet av pågående arbeid
	Synkrone kommunikasjonskanaler	[*] Ansikt-til-ansikt [*] Lynmeldinger	Diagrammer	[*] Burndown
	Asynkrone kommunikasjonskanaler [*] Epost	[*] Kommunikasjons-effektivitet: Varm eller Kald		
	Milepæler	[S] Oppdeling av sprinter		
Samlokalisert utvikling [Husk erfaring]	Samlokasert utviklerlag	[*] Ansikt-til-ansikt, [X] Parprogrammering	Valg av verktøy på bakgrunn av sam/delokasert utviklerlag	Personer og samspill fremfor prosesser og verktøy
	Hierarki for prosjektet	[S] Scrum of scrums		
Visualisering 00:40 [Husk erfaring]	Møtevirksomhet / avtaler	[*] Stand-up-møte	Møtevirksomhet	[S] Sprint Planning Meeting, Daily Scrum, Sprint Review Meeting, Sprint Retrospective
	Ikoner og grafikk fremfor tekst	[*] Whiteboard + noteIt		
	Oppgaver i arbeid	[K] CONWIP, [S] Arbeidsflyt mellom avdelinger / personer	Visualisering av tilgjengelig ressurser til enhver tid. Hvem	[*] Flaskehalser [L] Scope, ressurser, tid, kvalitet

			har tilgang?	
	Integrasjon av IDE	[*] Testing og problem-sporing tettere mot utviklere.	FishEye Greenhopper Confluence	[*] Bredere visualisering
	Visjon	[X] Metafor	Grafisk visualisert i verktøyet, eller faktisk design i verktøyet?	
Verktøy og opplæring *Ikke opp mot metoder!*	Innspill på valg av verktøy		Valgtakere av verktøy?	[*] Inkludere alle i laget
	Kjennskap til verktøy		Verktøy-kombinasjon	
	Effektivitet		Tilstrekkelig funksjonalitet	
	Trade-off			
	Tidsbruk av verktøy	[*] Personer og samspill fremfor prosesser og verktøy		
	Erfaring med andre prosjektverktøy	Hva fungerte bra? Hva fungerte mindre bra?	Tildeling av tilgang til prosjektverktøy til kunde?	**Samarbeid med kunde** fremfor kontraktsforhandlinger
Annet				

Avslutning

- Har du eksempler på at verktøyet har vært en avgjørende faktor for suksess eller fiasko?
- Har du opplevd at innhold i prosjektverktøyet har blitt lagt til for tidlig eller for sent i prosessen, som igjen kunne ha medført feil avgjørelser underveis i prosjektet?
- Andre opplevelser som f.eks «skrekk»-scenarioer?
- Er det andre ting du ønsker å legge til?